gd

GAME DEVELOPER MAGAZINE

JUNE 1998

UBM Tech

The DEER HUNTER Phenomenon

Following trends in game development is great sport whether you're inside or outside the industry, and with E3 upon us again, prognosticating which games will be hits next Christmas is the order of the day. One trend that's beginning to affect game development is the impact of the sub-$1,000 PC on game sales.

In 1996, the average price in the home PC market went under $1,500 for the first time. Compaq's Presario led the charge, offering sub-$1,000 systems based on Cyrix's MediaGX that often came in sealed cases that had no upgrade capabilities. Since their debut, low-priced systems (also called "commodity PCs" or "basic PCs") have become standard fare at consumer electronic stores, mass-market retail chains, and in the mail-order channel.

A sizeable portion of basic-PC purchases come from first-time buyers who now find they can afford a computer. An IDC/A.C. Nielson study last Christmas estimated a total of 1.5 million new PCs would be sold during that holiday season, 48 percent of which were to first-time buyers. Are these new PC owners different than the traditional PC owners? Perhaps so. Back in May 1997, *Marketing Computers* (a magazine worth reading if you want to follow trends in system sales) identified an emerging two-tiered marketing strategy by PC manufacturers: middle-income households were being split into upper- and lower-income categories and were being targeted differently. If systems manufacturers are approaching this middle-income market in a new way, it begs the question , should game publishers as well?

This may already be happening. As average PC prices have plunged, average PC game prices have come down as well. I did some quick calculations based on PC Data games sales numbers going back to 1995. Here's what I found: in 1995, the average price of the top 20 PC games for that year was $42.30. In 1996, that figure dropped 3.4 percent to $40.85. In 1997, it dropped 6.4 percent further to $38.25. For the latest month's results available as we go to press (February 1998), the average game price in that same list was $35.90.

Interestingly, as I scan last February's list (and many of the preceding months), the make up of the best seller list seems to be evolving as well. Last month I stated that the typical RPGs and action games based on science fiction and horror themes that were so prevalent in the best-seller list two years ago seem to be in decline, while titles based on licensed content from the world of children's toys are on the rise. There seems to be another game development trend that's shaping up as well. It's a shift toward content that, in the words of author Ben Sawyer, "enhances the pre-existing interests of consumers." It's based on the premise that if a gamer enjoys an activity in the real world (such as golf, snow boarding, or baseball), he or she might be inclined to buy a game based on that activity.

Sunstorm/GT Interactive's DEER HUNTER is an example of leveraging pre-existing interests. Its success, however, may also be related to the rise of the basic PC. Perhaps fantasy role-playing games, a traditional staple game genre, are too esoteric to a new tier of blue-collar consumers that are buying their first system? Maybe the steep hardware requirements of these games make a title like DEER HUNTER (which only requires a Pentium-75MHz and 30MB of hard drive space) look inviting. Certainly DEER HUNTER's $20 price tag helps sales if you're trying to attract a consumer who's very price-sensitive to begin with. Whatever the reason behind DEER HUNTER's success, this market has apparently been identified as low-hanging fruit by other developers: games based on trap and skeet shooting are reportedly on the way, as well as games about duck and geese hunting.

Where this is leading is still a big question mark. But it should give you pause. If the demographics of PC owners shifts towards those who are more price sensitive and less interested in traditional sci-fi/fantasy game themes, is there an opportunity there for you? ■

ON THE FRONT LINE OF GAME INNOVATION
Game DEVELOPER

EDITOR IN CHIEF Alex Dunne
adunne@compuserve.com

MANAGING EDITOR Tor Berg
tdberg@sirius.com

EDITORIAL ASSISTANT Wesley Hall
whall@mfi.com

ART DIRECTOR Laura Pool
lpool@mfi.com

EDITOR-AT-LARGE Chris Hecker
checker@d6.com

CONTRIBUTING EDITORS Jeff Lander
jeffl@darwin3d.com
Josh White
josh@vectorg.com
Omid Rahmat
omid@compuserve.com

ADVISORY BOARD Hal Barwood
Noah Falstein
Brian Hook
Susan Lee-Merrow
Mark Miller

COVER IMAGE Presto Studios

PUBLISHER Cynthia A. Blair
cblair@mfi.com

WESTERN REGIONAL SALES MANAGER Alicia Langer
(415) 905-2156
alanger@mfi.com

EASTERN REGIONAL SALES MANAGER Kim Love
(415) 905-2175
klove@mfi.com

SALES ASSOCIATE Ayrien Houchin
(415) 905-2788
ahouchin@mfi.com

MARKETING MANAGER Susan McDonald

AD. PRODUCTION COORDINATOR Dave Perrotti

DIRECTOR OF PRODUCTION Andrew A. Mickus

VICE PRESIDENT/CIRCULATION Jerry M. Okabe

ASST. CIRCULATION DIRECTOR Mike Poplardo

CIRCULATION MANAGER Stephanie Blake

CIRCULATION ASSISTANT Kausha Jackson-Craine

NEWSSTAND ANALYST Joyce Gorsuch

REPRINTS Stella Valdez
(916) 983-6971

un Miller Freeman
A United News & Media publication

CEO-MILLER FREEMAN GLOBAL Tony Tillin

CHAIRMAN-MILLER FREEMAN INC. Marshall W. Freeman

PRESIDENT/COO Donald A. Pazour

SENIOR VICE PRESIDENT/CFO Warren "Andy" Ambrose

SENIOR VICE PRESIDENTS H. Ted Bahr
Darrell Denny
David Nussbaum
Galen A. Poss
Wini D. Ragus
Regina Starr Ridley

VICE PRESIDENT/PRODUCTION Andrew A. Mickus

VICE PRESIDENT/CIRCULATION Jerry M. Okabe

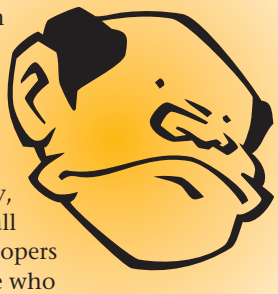
VICE PRESIDENT/SD SHOW GROUP KoAnn Vikören

SENIOR VICE PRESIDENT/SYSTEMS AND SOFTWARE DIVISION Regina Starr Ridley

www.gdmag.com

Goodbye, mTropolis

As a game developer and long-time mTropolis devotee, I'm extremely unhappy about Quark's decision to pull the plug on the product. We have over a thousand hours of work in our game, THE FORGOTTEN, all done in mTropolis, and we aren't the only ones. While we continue to work with ex-mFactory employees and third-party developers so that we can continue to use mTropolis for our title development, the abandoment of mTropolis — if the product only sits on the shelf — is not just bad for the mTropolis community, but for small game developers everywhere who have a great idea and a computer, but not the budget to hire a raft of C++ programmers or 20 seats of Authorware.



When I started using mTropolis, I was amazed. I said, "This is what a multimedia authoring environment should be." And I'm not alone. Anybody who's been a dedicated user of the product has had the same epiphany — and developing titles the mTropolis way, they wouldn't use any other approach. It's faster, more powerful, more productive, and more elegant. It inspired pioneering, dedication, and loyalty from third parties and end-users. We've worked on large projects for high-profile clients, as well as our own game title, and mTropolis has always performed like a champ. It's a reliable, robust, mature product that deserves a chance — it's the most perfect multimedia and game title development environment I've ever seen.

I beg Quark to reconsider killing mTropolis. If Quark will not or cannot market, maintain, sell, and update mTropolis as a product, I ask them to please consider a few other options:

1. Making it public domain, with ex-mFactory folk and the general mTropolis community responsible for future features and support. Quark, of course, would still hold all patent rights so that no competitor could use the technology. But the existing user base of this excellent, visionary product should not be abandoned.

2. Releasing it to the ex-mFactory people to maintain and update in their spare time as they work at their new jobs. Again, Quark retains all patent rights so there is no threat of competition, and the existing mTropolis community can continue to use it.

3. Put it up for sale or license. Terms of sale could include Quark holding onto the patents, but would allow someone to update, maintain, and make a small profit on the mTropolis product.

Solve the world's problems. E-mail us at gdmag@mfi.com. Or write to *Game Developer*, 600 Harrison Street, San Francisco, CA 94107.

4. Make it available free on-line with PDF documentation and a general, work-everywhere serial number. This supports both the existing mTropolis community and the dedicated third-party developers making mTools and MODs and other products for mTropolis 2.0. I will donate server space for this. Others, no doubt, would also donate server space — especially the third party developers whose livelihoods have become dependent on mTropolis. Quark holds all the patents, so no one would be making any damaging changes or additions.

5. Market it directly on-line for a low-price — say $299 or $199 — with PDF-only documentation, and see how it does. If the market grows large enough, it might just be profitable.

If these ideas won't work, I would appreciate Quark letting us know what the problems are so I, and the larger mTropolis community, can work to solve them in a productive way that will work for everybody. mTropolis is a superior environment, a great productivity tool, and a visionary product. I want to do whatever I can to make sure that it doesn't just die — and that it's available to the greatest number of people possible. No solution would make me as happy as Quark developing and aggressively marketing the product — but if that can't happen, I would like to see anything other than putting the greatest title development environment on the planet on the shelf. There have to be better alternatives — for Quark, the third parties, and the mTropolis and game developer communities.

Kevin S. Willis
Ransom Interactive

Using Velocity 128 with MAX 2

I was just reading the March 1998 issue of *Game Developer* and saw Josh White's review of MAX 2. In the article, he mentioned problems getting the STB Velocity 128 running with MAX 2. I've been experimenting with the same card, and here's what I found:

Using DirectX: Your display needs to be set to either 640×480 or 800×600 high color (the card will only work in 16 bits). You also need the DirectX 5 drivers. We're modeling baseball stadiums with 20+ textures on meshes under 1,000 polygons, and it zooms.

Using OpenGL: Download the beta OpenGL drivers for STB. Your display needs to be set to 640×480 in high color. (OpenGL uses the current display color depth, and the STB can only handle 16-bit.)

In the summer, a RIVA chipset will be released allowing access to 8MB — so more people can use it with MAX 2.

I like the idea of using a consumer-level graphics board for art and animation production. In the game industry, we need to be able to run game products as well as production tools. Most high-end boards won't run games, primarily because of their limited driver support.

As for the MAX 2 review, I agree with White's assessment of the product's UI. I hate the idea of an interface that only gives me access to vertices, faces, or edges on one object at a time. In MAX 2, you need to access an object, then it's modifer, then it's subobject before you can move one vertex. All I want is to grab a vertex and move it without entering the decathalon event of the mouse olympics.

This system does not work for artists. If I was sculpting in clay, I could just pinch and pull without any extra hassle. I have immediate access to every part of the clay. All of the other 3D programs have that immediate access too. Why not MAX?

Cyrus Lum
via e-mail

INDUSTRY WATCH

by Alex Dunne

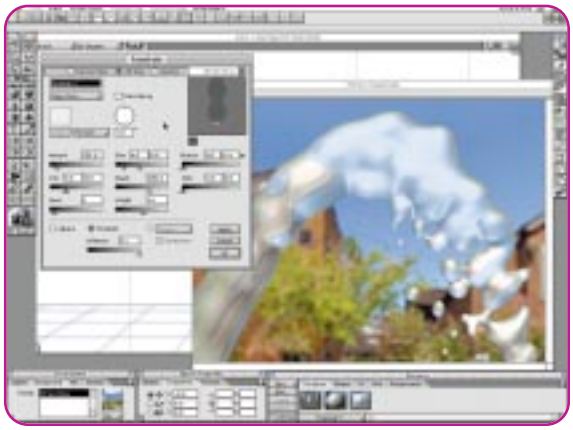
JAPANESE PREVIEW OAA MACHINE. LBE Systems, a U.S. arcade game developer, recently showed off a prototype of a Pentium II-based arcade game machine in Japan at the Tokyo Game Show '98. The machine, based on a 333MHz Pentium II, was running QUAKE II ARCADE EDITION. It was the first demonstration of Intel's Open Arcade Architecture in Japan.

MTROPOLIS KILLED. In a blow to its devoted but admittedly small user base, Quark Inc. announced that it was discontinuing its mTropolis multimedia development tool. The company acquired mFactory a year ago, and proceeded to upgrade the product to version 2.0, which it just shipped in March. In a strange, bittersweet announcement, the company said that it was both shipping the product for free to registered mTropolis customers, as well as discontinuing it. Quark representatives stated that mFactory technologies would be incorporated into future Quark multimedia products and may be licensed to other companies.

NIHILISTS UNITE! A new game development company, Nihilistic Software, was recently formed by industry veterans Rob Huebner, Ray Gresko, and Steve Tietze. Nihilistic will develop a 3D RPG as its first title, and Activision has agreed to publish it and two more of company's upcoming games. Gresko and Huebner previously rubbed shoulders at LucasArts as lead and senior programmers, respectively, on DARK FORCES II: JEDI KNIGHT. Tietze is an game artist and level designer who worked for Rogue Entertainment on QUAKE MISSION PACK #2, DISSOLUTION OF ETERNITY.

AND IN THIS CORNER... Here's a quick look at the legal fight card this

StudioPro Adds Windows



STRATA INC. has announced version 2.5 of its StudioPro 3D modeling, rendering, and animation program. The big news with this release is that StudioPro now runs under Windows in addition to its traditional Macintosh environment.

Enhancements with this version include multiprocessing support, network rendering, and velocity graphs. With support for Windows 95 and Windows NT (and soon, Windows 98), StudioPro now supports OpenGL. If you're creating type effects, StudioPro supports Postscript and TrueType fonts. Animation enhancements include four types of path splines, individual frame visualization, and time-varying polymesh. Strata has also added collision detection for particle effects such as PixieDust and Fountains.

And, of course, StudioPro is extensible, as any good 3D package should be. Strata is planning a whole series of plug-in Power Modules, and for a limited time, is offering Power Module 1 for free with the purchase of StudioPro 2.5. Power Module 1 includes effects such as Deform, Smooth, Mirror, Hair, 3D Fire&Smoke, 3D HotSpot tools, 3D PixieDust, and bones-based inverse kinematics.

StudioPro 2.5 has a suggested retail price of $1,495. The upgrade price from any 2.x version of StudioPro is $99, and Strata is offering a competitive upgrade from select other 3D packages for $599.

■ Strata Inc.
St. George, Utah
(800) 782-8233 / (435) 652-5221
www.strata.com

DirectX 6 Imminent

MICROSOFT, after beginning a general beta at its CGDC Seminar Day last May 9, is planning on releasing DirectX 6 for Windows 95 in July, 1998. DirectX 6 will also be available for Windows 98 and included in the final release of Windows NT 5.

While details at press time were still sketchy, Direct3D 6's planned feature set will reportedly include: support for multiple textures, new rasterizers, a geometry pipeline features, the DrawPrimitives2 DDI, a texture memory manager, flexible vertex format, vertex buffers, bump mapping, standard fixed-rate texture compression, opaque texture surfaces, alpha in texture palettes, luminance, stencil planes, W-buffering, and Z-buffer clearing.

However, word from the DirectX Group is that, in order to minimize risk to the DirectX 6 schedule, enhancements to DirectDraw have been limited to those that support new Direct3D features, plus the following: full–screen-only per-channel gamma control, the motion compensation DDI, and hardware de-interlacing support.

Microsoft is recommending to hardware manufacturers that they continue to improve and refine their DirectX 5 drivers, while adding new DirectX 6 functionality for release in drivers later

ASTS

OF GAME DEVELOPMENT

this year. Developers planning Christmas titles should continue developing their applications to the existing DirectX APIs, while planning DirectX 6 features as soon as they receive the beta.

- Microsoft Corp.
  Redmond, Wash.
  www.microsoft.com/directx/

New Miles

RAD GAME TOOLS INC. has released its Miles Sound System 4.0. This new version includes full DLS-1 MIDI support and integrated ADPCM compression support.

New DLS features include an integrated software synthesizer (with optional MMX support), support for S3 hardware DLS cards, tools to compress DLS sample files, tools to extract specific instruments from a generic DLS file, and tools to merge MIDI files with instruments into one song file.

The addition of IMA ADPCM compression support offers sound designers high-quality audio with four-to-one data compression. Plus, the Miles Sound System decompresses ADPCM data on the fly in its digital sound mixer.

Miles includes the new Sound Player, a freeware application that can play MIDI, XMIDI, DLS, compressed-DLS, and .WAV files. Miles also includes the Sound Studio utility, which can examine MIDI files, convert MIDI to XMIDI files, compress and expand DLS files, examine DLS files, compress and expand .WAV files, and merge DLS files with XMIDI files.

The Miles Sound Player and the Miles Sound Studio can both be downloaded from RAD's web site. The Miles Sound System SDK is available for Windows and is priced at $3,000 per title or $7,500 per site.

- RAD Game Tools
  Salt Lake City, Utah
  (801) 322-4200
  www.radgametools.com

Acid

SONIC FOUNDRY has expanded its already impressive line of audio processing tools with the release of Acid, a digital audio loop sequencer.

Acid automatically matches the tempo and pitch of the loop for instant synchronization. Users can change the pitch of a project by simply selecting the desired key from a drop-down menu. A beats-per-minute slider adjusts the tempo, and a the included tempo map can slow down or speed up loops.

Acid supports as many tracks as your system RAM can handle. Each track includes the volume, pan, and effect envelopes for greater editing control. Acid features DirectX audio plug-in support for incorporating real-time effects, as well as quick access to Sonic Foundry's Sound Forge and other audio editors. Acid is flexible enough to work with other software products and external hardware because it generates and chases SMPTE time code.

The product also ships with hundreds of prerecorded loops and will import 16- and 24-bit .WAV and .AIFF files from other sources. Non-looping, or one-shot, sounds are also supported. Acid can output .WAV or .AIFF files, or can export to Sonic Foundry's CD Architect or any other audio CD program.

Acid is available for Windows 95 and Windows NT and requires a Pentium 133MHz with a Windows-compatible sound card. It has a suggested retail price of $399.

- Sonic Foundry
  Madison, Wis.
  (608) 256-3133
  www.sonicfoundry.com

Errata

The Bit Blasts section of the April 1998 issue contained an announcement of The Duck Corp.'s TrueMotion 2.0. The phone number listed for Duck was incorrect. The correct phone number is (212) 941-2400.

month. In our first bout, we have Creative Labs taking on Aureal over the latter's Vortex chips. The lawsuit seeks "injunctive relief and damages" for alleged violations of a patent issued to Creative's subsidiary, E-mu. A second sparring match involves Creative yet again, which filed a patent infringement lawsuit against Diamond Multimedia and ESS Technology, alleging violation of a patent issued to E-mu. The lawsuit relates to PCI audio technology embodied in Diamond's Sonic Impact sound cards and ESS's Maestro-2 audio chip. And in our main event, SGI is suing nVidia to block the company from making its Riva processors. The lawsuit charges that the Riva processors infringe upon SGI's patent covering high-speed texture mapping in low-priced hardware. The lawsuit seeks unspecified damages. nVidia said it would vigorously fight the lawsuit, but that's got to hurt in light of the fact that nVidia is in the process of going public right now.

WHEN IS A LOSS GOOD NEWS? **When the loss isn't as bad as expected. Witness Acclaim, which reported a a net loss of $1.2 million on net revenues of $69.3 million for its second fiscal quarter, compared with a net loss of $16.8 million on net revenues of $52.3 million in the comparable quarter of 1997. The company's uptick was due to strong sales of N64 titles such as NFL QUARTERBACK CLUB '98, TUROK: DINOSAUR HUNTER, NHL BREAKAWAY '98, and RIVEN. The company said N64 games generated 60 percent of sales during the quarter.**

BRODERBUMMED. **Broderbund laid off 70 employees, representing about seven percent of its workforce, in a move driven by the need to trim some fat from the company's ranks. Most of those pink slipped worked in the product development and product marketing divisions. The company revealed that the axe may fall again soon, as it is looking for ways to eliminate $5 million in spending per year. Broderbund recently posted earnings of $3.9 million, up from $3.2 million for the same quarter a year ago.**

Picking a Pack of Pixels

Much of the technology I develop goes into the support of a game project and is never seen by the game player. This technology takes the form of the game development tools actually used to create the game, rather than play it. Many times, the tool development on a game project takes much longer than the technical creation of the game itself. This may seem like a waste of time, but creating effective and easy-to-use production tools is the single greatest thing you can do to speed up your project. I like to think that for every week I spend speeding up the production pathway by creating the right tools for the project, I'll save months of production work down the line.

Sometimes, this is a tough concept for management to grasp, especially because those early days of tool development don't yield much in the way of sexy demos. In fact, boring things with no sales appeal at all, such as user interfaces, can be very difficult to get right. But, if you take your time and solicit input from the people who'll be using the tools, things will work out better. Also, your level designers will love you for it.

This brings me to this month's topic: user interfaces. Last month, I demonstrated skeletal deformation of a single mesh ("Skin Them Bones: Game Programming for the Web Generation," May 1998). These sorts of demo projects bring up a real problem. For these demo projects, I want to focus on the specific task that I'm demonstrating. But, in order to get the minimum functionality to demonstrate the concept, I need to create quite a bit of structural code. I end up writing hundreds of lines of code just to demonstrate one routine effectively. However, as graphics programming topics get more and more complex, this is a problem we all just have to live with.

This specific skeletal-deformation-of-a-single-mesh demo required that the user be able to select vertices on the mesh object so the weights for these vertices could be adjusted. Sure, that's easy to say, but I need to translate it into code. So what do I really need?

Pick Box

As the programmer, I want to create a selection box when the user holds the [shift] key and drags the mouse. Depending on whether the user holds down the left or right mouse button, I want to select or deselect vertices corresponding to this boxed-in area. I'm going to call this graphic effect a pick box. A pick box is shown in Figure 1. In the Win32 GDI, this would be a pretty easy task. I can track where the user clicked the mouse and draw a rectangle with the GDI command **FrameRect**. However, I want to use OpenGL because the rest of my render is in OpenGL and the code will be cleaner. The other, more important reason for employing OpenGL is cross-platform compatibility. I can easily convert this tool over to the SGI or another OpenGL platform. It's also accelerated by OpenGL hardware that supports hardware rendering of line primitives.

So, how do I go about doing this in OpenGL? You may think that it would be as easy as drawing some 2D OpenGL primitive lines at the end of my normal rendering loop. I could accomplish this with some code that looked like Listing 1.

The problem with this plan is that my main render window is in perspective. The call to place a 2D vertex with **glVertex2s** assumes that z = 0. So this code would draw a rectangle in world space in the plane where z = 0. Once perspective is taken into account, these lines will be nowhere near where I want them on the screen.

I really want to draw those 2D lines in screen space instead of 3D world space. Fortunately for me, OpenGL has a very easy method for doing just this. I'll need

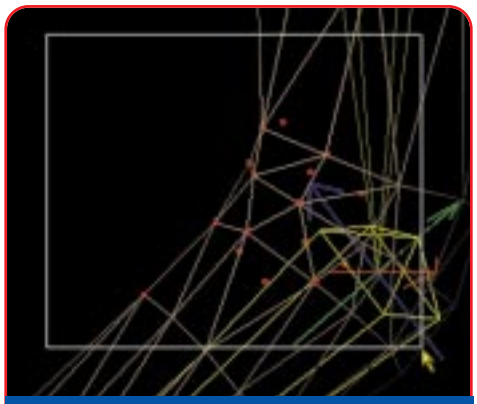


FIGURE 1. *A sample pick box.*

LISTING 1. *2D OpenGL primitive lines.*

```
glBegin(GL_LINE_STRIP);
        glVertex2s((short)m_SelectRect.left,(short)m_SelectRect.top);
        glVertex2s((short)m_SelectRect.right,(short)m_SelectRect.top);
        glVertex2s((short)m_SelectRect.right,(short)m_SelectRect.bottom);
        glVertex2s((short)m_SelectRect.left,(short)m_SelectRect.bottom);
        glVertex2s((short)m_SelectRect.left,(short)m_SelectRect.top);
        glEnd();
```

When not bending the bones of some strange alien creature, Jeff can be found hanging out at his studio at the beach. See if you can smack some sense into him by writing to jeffl@darwin3d.com.

to change my perspective view to an orthographic parallel projection where the value for z won't affect the image. However, I need to save my perspective projection so I can go back to it after I'm finished selecting. I do this by manipulating the Projection stack much in the way that I manipulated the ModelView stack for my animation. I can push my current projection matrix onto the stack and then play around with it. The code to do this is in Listing 2.

I first put OpenGL in Projection mode with a call to **glMatrixMode**, then I call **glPushMatrix**. This saves the projection so that I can restore it later. I load in a new matrix with the call to **glLoadIdentity**, then the call to **gluOrtho2D** creates the new projection matrix with the screen size information. This way, the pixels match the window settings exactly. One thing you need to remember is that OpenGL considers y = 0 to be at the bottom of the screen. This isn't the way that the Windows screen handles coordinates, and you'll need to adjust accordingly.

With all this work out of the way, the problem becomes simply drawing the 2D lines. I can quickly accomplish this with a series of **GL_LINE_STRIP** coordinates. At the end of the routine, the matrix is popped back and the MatrixMode is set back to **GL_MODELVIEW**, just to be friendly.

Now that I have my nice, hardware-accelerated pick box, I need to know how to detect what's inside of it. That brings us to...

Feedback

Now, I'm not talking about the feedback you get when your guitar gets too close to the amp or, more likely if you're reading this, when your voice recognition microphone gets too close to your PC speakers. I'm talking about the kind of feedback by which OpenGL lets you know what it's doing behind the hardware curtain.

You see, when I was dealing with my own software renderer, it was very easy to find out what was going on. I could find out where the vertices were in camera space, screen space, texture space — whatever. Enter hardware. With 3D graphics hardware, all this information is abstracted away from me, the programmer. I just give the objects to the API in model space with their transformation information, and away it goes. Because any portion of this pipeline could potentially be accelerated by hardware, it's all hidden from my view. Now, because I created the projection matrix and know all the transformations, I could recreate the 3D math pipeline and get the information that I need. However, this is a lot of duplicate work and wouldn't allow for geometry hardware acceleration. There are times when I need to get in this deep, for example, when doing the deformations last month. Thankfully, however, this isn't one of those times.

OpenGL provides a mechanism for getting the results of its transformation operations. This mechanism is called feedback. You can put the system in feedback mode and submit

LISTING 2. *Drawing the pick box.*

```
glMatrixMode(GL_PROJECTION);                          // I WANT TO PLAY WITH THE PROJECTION
glPushMatrix();                                       // SAVE THE OLD ONE
     glLoadIdentity();                                // LOAD A NEW ONE
     gluOrtho2D(0,m_ScreenWidth,0,m_ScreenHeight); // USE WINDOW SETTINGS
     glColor3f(1.0f, 1.0f, 1.0f);                     // DRAW A WHITE BOX
     glBegin(GL_LINE_STRIP);
     glVertex2s((short)m_SelectRect.left,(short)m_SelectRect.top);
     glVertex2s((short)m_SelectRect.right,(short)m_SelectRect.top);
     glVertex2s((short)m_SelectRect.right,(short)m_SelectRect.bottom);
     glVertex2s((short)m_SelectRect.left,(short)m_SelectRect.bottom);
     glVertex2s((short)m_SelectRect.left,(short)m_SelectRect.top);
     glEnd();
glPopMatrix();                                        // RESTORE THE OLD PROJECTION
glMatrixMode(GL_MODELVIEW);                           // BACK TO MODEL MODE
```

LISTING 3. *Creating the Feedback buffer.*

```
////////////////////////////////////////////////////////////////////////////
// Function:    SelectVertices
// Purpose:     Use Feedback to get all the vertices in the view
// Arguments:   Should I select or de-select?
////////////////////////////////////////////////////////////////////////////
void COGLView::SelectVertices(BOOL select)
{
/// Local Variables ///////////////////////////////////////////////////////
     GLfloat *feedBuffer;
     GLint hitCount;
     tColoredVertex *meshdata; // IN THIS CASE THE DATA IS COLOR VERTICES
     int loop;
////////////////////////////////////////////////////////////////////////////
     // GET A TEMP POINTER TO THE ACTUAL VERTEX DATA
     meshdata = m_BaseMesh;
     // INITIALIZE A PLACE TO PUT ALL THE FEEDBACK INFO (3 DATA, 1 TAG, 2 TOKENS)
     feedBuffer = (GLfloat *)malloc(sizeof(GLfloat) * m_Mesh.desc->pointCnt * 6);
     // TELL OPENGL ABOUT THE BUFFER
     glFeedbackBuffer(m_Mesh.desc->pointCnt * 6,GL_3D,feedBuffer);
     (void)glRenderMode(GL_FEEDBACK);    // SET IT IN FEEDBACK MODE

     for (loop = 0; loop  < m_Mesh.desc->pointCnt; loop++)
     {
          // PASS THROUGH A MARKET LETTING ME KNOW WHAT VERTEX IT WAS
          glPassThrough((float)loop);
          // SEND THE VERTEX
          glBegin(GL_POINTS);
          glVertex3f(meshdata[loop].x,meshdata[loop].y,meshdata[loop].z);
          glEnd();
     }
     hitCount = glRenderMode(GL_RENDER); // HOW MANY HITS DID I GET
     CompareBuffer(hitCount,feedBuffer, select);
          // CHECK THEM AGAINST MY SELECTION
     free(feedBuffer);         // GET RID OF THE MEMORY
}
////// SelectVertices ////////////////////////////////////////////////////////
```

some buffer space to the renderer. OpenGL then fills this buffer with the information used in creating your display. Before you set OpenGL in feedback mode, you also need to tell the renderer about the buffer that it should fill with information. This is accomplished with a call to `glFeedbackBuffer (size, type, buffer);` where **size** is the length in **float**s of the buffer you provided, **type** is the format of the information you want, and **buffer** is a pointer to the actual buffer. In my application, I want the users to select the vertices needed for weighting. This means that I can represent all the vertices as `GL_POINT` primitives. For this type of application, the information type I'm interested in is OpenGL type `GL_3D`, as I just want the x, y, z coordinates of the transformed point. Next, I calculate the size of buffer that I'll need. Since I'm drawing all the vertices with `GL_POINT` and I've set the type to `GL_3D`, I'll be getting back four **float**s for each vertex. The extra **float** will be a token, a number defined in OpenGL to represent a operation type, which declares what the next set of data will be. I also want a method for knowing which original vertex is being drawn. If some portion of the model is off the screen, it will be clipped and not returned in the feedback buffer. If any vertex were clipped, I wouldn't be able to count on getting information for every vertex in order. This would make deciding what vertex was selected a lot more difficult.

So, I add what's called a `GL_PASS_THROUGH_TOKEN` for each vertex. The token is equal to the vertex number that I am currently drawing. This token will mark each vertex as it passes through to OpenGL, telling me which vertices ended up in the view. The addition of the pass-through token brings the total amount of data returned for each vertex to six **float**s. You can see the layout of that information in Table 1.

Now that all this set up work is finished, I can send all the vertices to the renderer and look through the buffer. The final code to set up and render the feedback buffer is in Listing 4. The final step in the feedback rendering process is to figure out how much data is in my buffer after the render is complete. The call to reset the render mode back to normal actually returns the number of data items placed in the buffer. I grab that number and pass it on to the routine that compares the buffer.

TABLE 1. *GL_PASS_THROUGH_TOKEN.*

vertex number
GL_POINT_TOKEN
x (transformed to screen space)
y (transformed to screen space)
z (transformed to screen space)

Checking Your Results

Once my feedback buffer is filled and I know how many items are in it, it's time to find out if any of these points are within my selection rectangle. It's a very straight forward march through the data buffer. When I get a `GL_PASS_THROUGH_TOKEN`, I save the data value as the current vertex. The next token

LISTING 4. *The final code to set up the Feedback buffer.*

```
// INITIALIZE A BUFFER (3 DATA, 1 TAG, 2 TOKENS) PER VERTEX
feedBuffer = (GLfloat *)malloc(sizeof(GLfloat) *
m_Mesh.desc->pointCnt * 6);
// TELL OPENGL ABOUT THE BUFFER
glFeedbackBuffer(m_Mesh.desc->pointCnt * 6,
GL_3D,feedBuffer);
```

The call to put OpenGL in feedback mode is `glRenderMode(GL_FEEDBACK);`

LISTING 5. *Checking if I selected anything.*

```
////////////////////////////////////////////////////////////////////////////
// Function:    CompareBuffer
// Purpose:     Check the feedback buffer to see if anything is tagged
// Arguments:   Number of hits, pointer to buffer, Should I select or de-select
////////////////////////////////////////////////////////////////////////////
void COGLView::CompareBuffer(GLint size, GLfloat *buffer,BOOL select)
{
/// Local Variables ///////////////////////////////////////////////////////
     GLint count;
     GLfloat token,point[3];
     int loop,currentVertex;
////////////////////////////////////////////////////////////////////////////
     count = size;
     while (count)
     {
          token = buffer[size - count];  // CHECK THE TOKEN
          count--;
          if (token == GL_PASS_THROUGH_TOKEN)      // VERTEX MARKER
          {
               currentVertex = (int)buffer[size - count]; // WHAT VERTEX
               count--;
          }
          else if (token == GL_POINT_TOKEN)
          {
               // THERE ARE THREE ELEMENTS TO A POINT TOKEN
               for (loop = 0; loop < 3; loop++)
               {
                    point[loop] = buffer[size - count];
                    count--;
               }
               // CHECK IF THE POINT WAS IN MY SELECTION RECTANGLE
               // FLOATS 0 AND 1 ARE SCREEN X AND Y
               // NOTE: OPENGL SETS THE BOTTOM Y=0
               if (point[0] >= m_SelectRect.left &&
                    point[0] <= m_SelectRect.right &&
                    point[1] <= m_SelectRect.top &&
                    point[1] >= m_SelectRect.bottom)
                    // SET THIS VERTEX TO THE CURRENT SELECTION VALUE
                    m_SelectFlags[currentVertex] = select;
          }
     }
}
////// CompareBuffer ////////////////////////////////////////////////////////
```

should be a **GL_POINT_TOKEN** with the x, y, z coordinates in screen space. I simply compare the x and y portions of this vertex to the selection rectangle, remembering that OpenGL considers y to be 0 at the bottom of the screen. If the transformed vertex is in the selection rectangle, I set the selection flag for that vertex according to the selection mode. You can see this routine in Listing 5.

So What Do I Have Now?

These techniques prove very effective in allowing a user to select specific vertices in a 3D model. While I use these OpenGL methods to create a user interface for a development tool, there are many other potential uses. It would be very easy to use the pick box and feedback method in a real-time strategy game to select units. In fact, any application that requires selection of an object in a 3D rendered window could benefit from methods such as these.

There is no specific demo this month. You can look at the source code and application from May 1998 to see how these things turned out. Get the goods on the *Game Developer* web site (www.gdmag.com). ■

REFERENCES

I mainly relied on the Red and Blue books for the information in this column. These are the main reference books for OpenGL, and I recommend them highly to anyone working in OpenGL.

Woo, Mason, Jackie Neider, and Tom Davis. *OpenGL Programming Guide: The Official Guide to Learning OpenGL*. Second Edition. Menlo Park, Calif.: Addison-Wesley Developers Press, 1997.

Kempf, Renate, and Chris Frazier (eds). *OpenGL Reference Manual: The Official Reference Document to OpenGL*. Second Edition. Menlo Park, Calif.: Addison-Wesley Developers Press, 1997.

The White book also has been useful to me, as it applies directly to OpenGL on Windows 95 and Windows NT.

Fosner, Ron. *OpenGL: Programming for Windows 95 and Windows NT*. Menlo Park, Calif.: Addison Wesley Developers Press, 1997.

Mea Culpa Code

Several of you have written to me pointing out that some of the commenting in the Quaternion SLERP code in April 1998 had it all wrong. The code in question handled a fairly rare and extremely specific case. The routine functioned as expected, but my commenting of the special case handling was the problem. I noticed the error myself after the issue was shipped. See the corrected code below.

The first comparison checks for a special case where the two quaternions are almost directly opposite within the angle threshold. If not, they are handled by either SLERP or LERP depending on the change in angle. If the quaternions were directly opposite, the sine of the angle between them would be zero, leading to a divided by zero error. So, the routine instead creates a perpendicular quaternion and interpolates through that one.

This is a rare case. In fact, I needed to force situations to test it out. I originally thought the test's purpose was to ensure travel by the slowest arc. The Watt and Watt book mentions this but doesn't implement it in code. I added code to do this, but found that really wasn't what I needed. In many situations, I wanted to travel by the exact arc I keyframed and not the shortest. I apologize for any confusion. The lesson here is to always experiment and create many sample conditions to test the limits of a routine. I'm happy that people followed what I was talking about enough to point out problems. I hope this is a learning process that we all gain from.

```
void SlerpQuat(tQuaternion *quat1,tQuaternion *quat2,float slerp, tQuaternion *result)
{
/// Local Variables ///////////////////////////////////////////////////////
     double omega,cosom,sinom,scale0,scale1;
////////////////////////////////////////////////////////////////////////////
     // USE THE DOT PRODUCT TO GET THE COSINE OF THE ANGLE BETWEEN THE QUATERNIONS
     cosom = quat1->x * quat2->x +
             quat1->y * quat2->y +
             quat1->z * quat2->z +
             quat1->w * quat2->w;

     // CHECK A COUPLE OF SPECIAL CASES.
     // MAKE SURE THE TWO QUATERNIONS ARE NOT EXACTLY OPPOSITE? (WITHIN A LITTLE SLOP)
     if ((1.0 + cosom) > DELTA)
     {
     // ARE THEY MORE THAN A LITTLE BIT DIFFERENT? AVOID A DIVIDED BY ZERO AND LERP IF NOT
          if ((1.0 - cosom) > DELTA) {
               // YES, DO A SLERP
               omega = acos(cosom);
               sinom = sin(omega);
               scale0 = sin((1.0 - slerp) * omega) / sinom;
               scale1 = sin(slerp * omega) / sinom;
          } else {
               // NOT A VERY BIG DIFFERENCE, DO A LERP
               scale0 = 1.0 - slerp;
               scale1 = slerp;
          }
          result->x = scale0 * quat1->x + scale1 * quat2->x;
          result->y = scale0 * quat1->y + scale1 * quat2->y;
          result->z = scale0 * quat1->z + scale1 * quat2->z;
          result->w = scale0 * quat1->w + scale1 * quat2->w;
     } else {
          // THE QUATERNIONS ARE NEARLY OPPOSITE SO TO AVOID A DIVIDED BY ZERO ERROR
          // CALCULATE A PERPENDICULAR QUATERNION AND SLERP THAT DIRECTION
          result->x = -quat2->y;
          result->y = quat2->x;
          result->z = -quat2->w;
          result->w = quat2->z;
          scale0 = sin((1.0 - slerp) * (float)HALF_PI);
          scale1 = sin(slerp * (float)HALF_PI);
          result->x = scale0 * quat1->x + scale1 * result->x;
          result->y = scale0 * quat1->y + scale1 * result->y;
          result->z = scale0 * quat1->z + scale1 * result->z;
          result->w = scale0 * quat1->w + scale1 * result->w;
     }
}
// SlerpQuat ////////////////////////////////////////////////////////////////
```

Oldtimer's Guide to Better Textures

ldtimer: Hell's bells, boy, them textures are lamer than my old sway-backed mule.

Whippersnapper: Well, gosh, Mr. Oldtimer, what can I do? I've only got 100K of texture memory, and I have to texture a whole building with it.

Oldtimer: That's no excuse, son. Back in my day, we had to hand-draw the polygons with graph paper and slide rules. Now, with all them power steering and chrome knobs for darn near everything, you young-uns are fat and lazy.

Whipper: Now hold on just a minute, Oldtimer. You're being impolite to my XG4000ProArtStation? Why, this thing's got AutoLOD, 16MB texture RAM, and chrome spark plugs.

Oldtimer: Boy, that thing couldn't even draw a Cathy cartoon by itself! It's just a machine. Real art is drawed by real people, not some clipart-snorting contraption.

Whipper: Hey now, I just spent four years slaving in the Superior Academy of Artwork Excellence, so don't tell me I don't know how to draw.

Oldtimer: Heh, heh. O.K., now hold them horses a minute. Ain't no doubt you can draw. I saw that nekkid sword-lady with the chainmail brassiere and all that you drawed. Hoowee!

Whipper: That's right. BondoGirl is a honey. I guess you were on the hiring committee, huh?

Oldtimer: Yes indeedy. I know you got talent, or you wouldn't even be here. Now, all you pups need is a little schooling in squishing all that talent into something that your graphics engine contraption can handle.

Whipper: Hmph. I bet modern times have passed you by, Oldtimer. What do you know?

Oldtimer: Skeptical, eh? O.K., take a look at this here Figure 1. That church only uses two 128×128 full-color textures — it ain't but 96K texture memory.

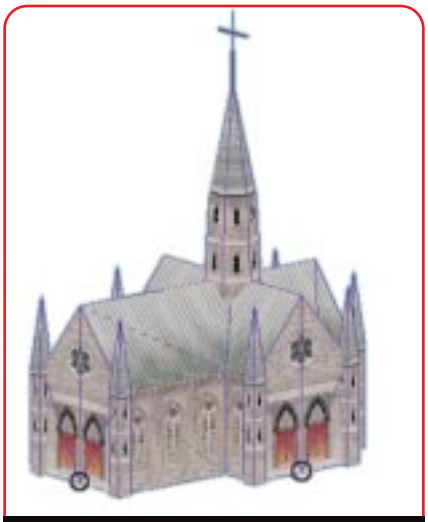


FIGURE 1. *Church model.*

Whipper: O.K., not bad. So how did you do it?

Oldtimer: Boy, I got so many different tricks, I don't even know where to start. I'd talk your leg off before I could explain all the stuff what goes into this model.

Whipper: Sure, pops. I bet you're going to tell me how to match palettes, like they did for World War II, right? Well, guess what. This is the modern age, and we don't use 8-bit palettes any more. All the new graphics engines have 16-bit color and bucket seats. I bet you don't know much that really matters anymore.

Oldtimer: Tarnation, you young things are uppity! You just set back and let the Oldtimer tell you how it's done. We'll just talk about texture maps today, since that's all your fresh little pea-brain could handle in one day.

Whipper: O.K., I'm ready.

Oldtimer's First Lesson: Use Symmetry When Possible

First off, see the two church doors in Figure 1? They look exactly like each other, right? Well, that's because they're the same texture, used twice. I just painted half the texture — saves a whole heap of memory right off the bat. Look at the textures in Figure 2 and you'll see what I mean.

It's especially easy to apply symmetric textures when the geometry is also symmetric, as is this church. That's because you can do the two steps at the same time. Build half the geometry, apply the texture and mapping to the half, mirror it, and you're done.

Also, this trick works best when the geometry doesn't touch the original, as with the towers of the church. When you're building something like the main body of the church or a head, things get tricky at the centerline. The hitch? You have to keep a few of the vertices on the mirror plane, even if the geometry doesn't use them. That's so the mapping coordinates get stored at the centerline.

Josh White runs Vector Graphics, a real-time 3D art production company. He wrote Designing 3D Graphics *(Wiley Computer Publishing, 1996), he has spoken at the CGDC, and he cofounded the CGA, an open association of computer game artists. You can reach him at column@vectorg.com.*

FIGURE 2. *Church texture 1.*

For example, you'd keep the vertices circled in Figure 1, even though they'd normally be useless because they don't define anything. But here, if you weld them up, you'd get a nasty texture stretch along the bottom. So reusing textures with symmetry usually adds a few extra vertices, but since it saves all that memory, most folks decide it's worthwhile.

"O.K., sure," quips young Whipper. "Trading a couple of vertices for a large memory savings makes sense. That worked pretty well here, but what if your model isn't symmetric?"

That's a problem all right, and usually, it can't be helped. But sometimes you can sweet-talk the designer (if you aren't designing the model yourself) into changing the design so you can get some symmetry reuse. Obviously, however, that's not practical for most asymmetric models.

If you can't use symmetry, try to think of other ways to reuse the textures. For example, maybe you can use the doors on the long side of the church, as well as on the front. This is usually a lot more difficult and time-consuming than simply building half, texturing, and mirroring, and it's often not worth your time unless your texture budgets (you do know how much texture memory you're allowed to use, don't you?) are really tight.

Oldtimer's Second Lesson: Use the Lowest Color Depth Possible

This is a simple one: if your graphics engine can handle them, save most of your textures in a simple color format. For example, instead of saving your textures in 24-bit, use 16- or 8-bit.

Of course, artists love 24-bit because it gives millions of colors, no banding, and so on. Why would you give that up? When you don't need it, 24-bit color is often wasteful. Figure 3 is intended to help you understand how much data is used by an image: you have to know dimensions and color depth to know the memory usage. The 32-bit image (24-bit plus 8 bits of alpha) on the left is using four times the memory of the 8-bit image on the right. For textures such as this, it's completely overkill. There aren't more than 256 unique colors in the texture, so the 8-bit image will look exactly as good as a 24-bit image. There isn't any masking or usage of the alpha channel either, so that's a total waste as well.

Still not sold on the idea of reducing color depth? Think of it this way: you could either store a single 24-bit wall, or three 8-bit walls — one plain, one with a window, and one with a dirt smudge. Which option will give you a better-looking model at the end of the day? I'll bet you'd rather have some windows and dirty parts. Most textures don't look bad in 8-bit... and if you do find one that curdles the eyeballs, you can always keep that one as a 16- or 24-bit image.

But, there's a major caveat. "If the graphics engine can handle them" is, right now, a really common limitation. In a normal 1998 game, it's rare that the graphics engine will store an 8-bit texture. Usually, 8-bit textures are converted to 16- or 24-bit when the graph-

Unsupported Modeling Tool Features

Whipper: My XG4oooModel-O-Matic software supports texture cropping. It'll let me tile a piece of a texture map instead of using the whole thing.

Oldtimer: Heh heh... you young pups. Son, you ever tried exporting that tiled object into any graphics engine — even a rusty old VRML browser?

Whipper: Well, not exactly... You suggesting that it wouldn't work? Why not?

Oldtimer: Darn tooting it wouldn't, unless one-a-them grease monkey programmers wrote a special handler for it, and maybe not even then. Problem is, that tiling assumption is built into the actual graphics hardware of your hotrod 3D graphics card — the chip itself does the tiling from the edges of the bitmap.

Whipper: ...oh.

Oldtimer: There's your first big lesson. Don't never think nothing in your 3D modeling program will work in the graphics engine, unless you know for a fact that it does.

Whipper: Yeah, yeah, O.K., what else you got to tell?

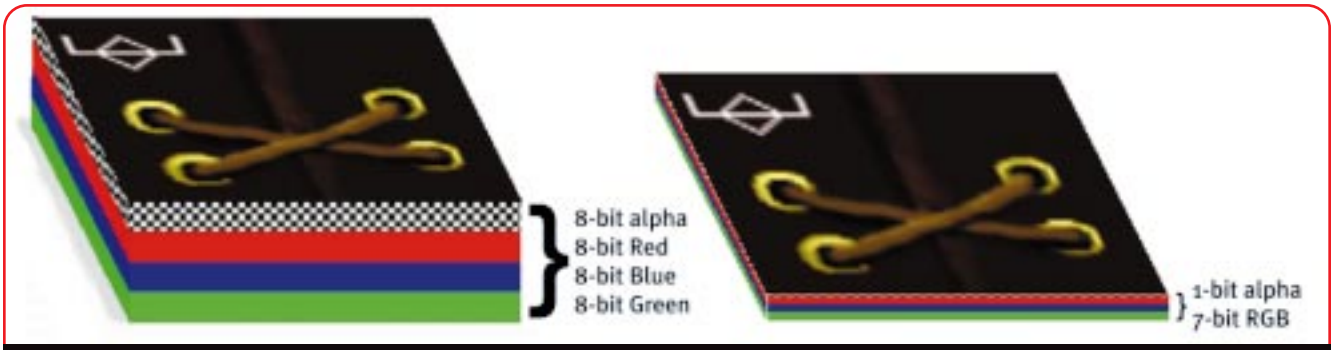


FIGURE 3. *Different color depths.*

ics engine loads them. That's the worst of both worlds — all the banding of an 8-bit image, with all the memory usage of a 24-bit image.

Native support for 8-bit images is an important feature to ask your programmers for, so let me say it loud and clear: RT3D engines and hardware should have full support for textures with multiple color depths.

Oldtimer's Third Lesson: Fighting the Power-of-Two Texture Sizing

Take a look back at Figure 2, one of the two textures on the church. Now look back at the church in Figure 1, especially where the tower meets the main building. It's hard to see, so look closely. You'll see that sloping-roof part is used separately from the tower part. That means what we really have here is a couple of textures clumped into one image.

Of course, it's harder to map that way, but I didn't just make two separate textures for no reason. We're fighting the limits of that graphics engine again. This time, the rule is to use textures that are a power of two in size.

We separate the two textures on the model by placing the mapping coordinates so that each texture is used where it belongs on the model, like the front of the church in Figure 5. The big yellow lines show three vertices in mesh and the UV coordinates of those vertices on the texture. The point is that the front faces aren't mapped to the edge of the texture.

This trick isn't very convenient. It takes more time to hassle with projection mapping icons, lining everything up right. Still, if you want the best use of your texture memory, it's what you have to do. Another problem, while we're thinking about drawbacks, is that you can't tile the texture with most graphics engines, because they assume that the mapping goes all the way to the edge when it wraps around.

Oldtimer's Fourth Lesson: Using Stretch/Streaking

You may already know this one, but if you have a texture that naturally streaks, such as the roof of the church, it still looks O.K. if you squish it in the streaking direction, then stretch it back out to the original size with mapping (Figure 6).

The lower row of roofs are images of the textures above them, mapped stretched-out so that they all fit the same polygon. On the left is the original, which looks same above and below. In the middle, the top image is squished 50 percent, but you can hardly tell in the bottom. On the right, the poor texture was squished 75 percent, which is to say it lost three pixels out of four. You can start to see banding, but it's not really that bad. Actually, it's darn near O.K. where the color doesn't change so much at the top of the image. It's a small trick, but the idea is to build yourself a toolbox of little tricks to rummage through when you're stuck.

The Power of Two Thing

Whipper: Power of two? What's that?

Oldtimer: Well, I'll be. Son, a power of two is when you keep doubling two: 2, 4, 8, 16, 32, 64, 128, 256, 512, and on and on. You'll see them numbers everywhere in computers if you look.

Whipper: That's so stupid! Powers of ten make more sense to me.

Oldtimer: See, now powers of two are as easy as powers of ten for the computer. In base two, which is how them rattletraps work, you get to the next power of two by adding a zero, just like we do with 10 to 100. Computers can handle these powers of two right quick-like.

Whipper: Huh, O.K.

Oldtimer: Anyway, so we got to make all textures powers of two in size. See how each image is a quarter of the next in Figure 4?

Whipper: Yeah, they're a big jump in quality.

Oldtimer: Danged right it's a big jump. Each different size uses four times the memory of the previous one. Say we got the front texture of the church and it's 200 pixels wide. We don't need a 256x256 texture, but we want something better than 128x128. What'll we do?

Whipper: Ah, I see. That's when we put two images in the same texture — like you were saying before we got out in this sidebar — to fill out the 256x256 texture.

Oldtimer: Yeehaw! That's right. Now let's get back to the main lesson body.

FIGURE 4. *Texture scaled by powers of two.*

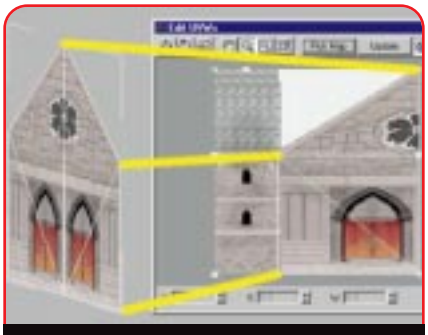


FIGURE 5. *Partially mapping a texture.*

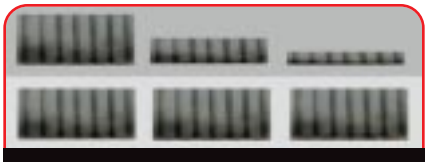


FIGURE 6. *Stretched textures can look O.K. for certain designs.*

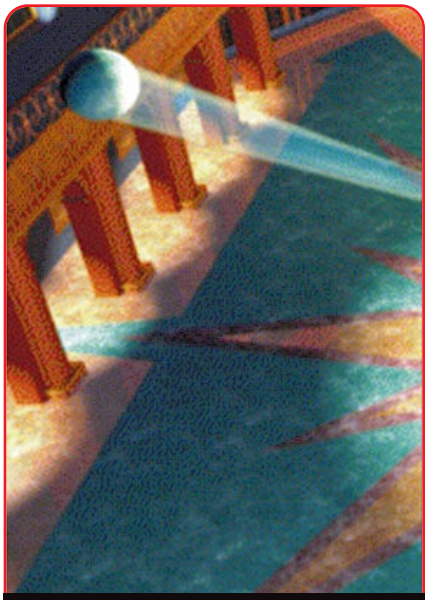


FIGURE 7. *Ballroom floor.*

Oldtimer's Fifth Lesson: Use Tiling of Small Textures Where Possible

Do you hate tiling? You can't abide all those repeating patterns, right? You've seen too many grasslands that look like green linoleum?

There's a whole art to making repeating patterns that don't stand up and bite you. Although tiled textures always looks worse than untiled, sometimes tiling is the only thing you can do — there just isn't enough texture memory. Anyway, you can get nice results with tiling — look at the floor in Figure 7.

"That's not tiled!" protests Whipper. "There's gradient light on it... or is that QUAKE lighting?" It's a tiled image with a blended lightmap over it. Take a look at the components in Figure 8.

What you can't tell from the images is how much memory that saved. That floor is a 2,000×1,000-pixel area, with just four 64×64 textures. It's using three 16-bit marble textures that are 64 pixels square on the left: one green, one brown, and one red (just on the edge of the starburst). Then there's the lightmap, which is 64×128, but only 8-bit grayscale, so it uses about the same memory as a 64×64 color texture. So it used 8,192 bytes per each of the four textures, for a total of 32K texture memory. Compare that to a worst-case 4,194K texture memory to cover that area in 16 bits without tiling.

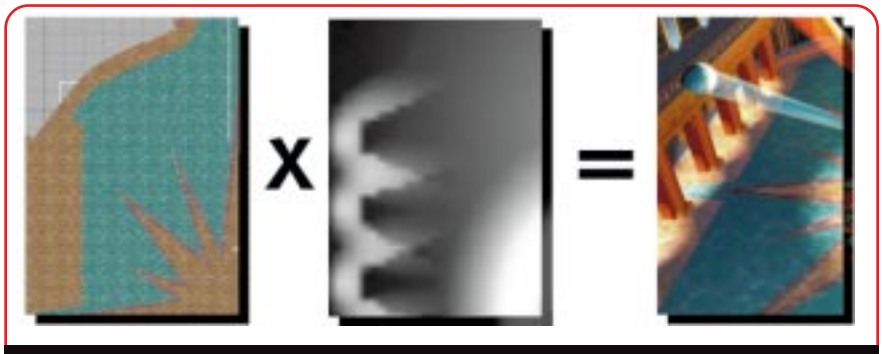


FIGURE 8. *Ballroom floor construction.*

Finale

Whipper: Wow, that's pretty darned good.

Oldtimer: Why thank you, son. Of course, the reason it works so well is texture blending, combining the lightmap with the floor textures. Turns out there's all sorts of fancy new ways to do that. Like see here...

Whipper: Err, unfortunately, I'm, uh, late for an appointment. Perhaps tomorrow you can come by and show me what you know about texture blending, O.K.?

Oldtimer: O.K., go run off and leave me here. Maybe I'll see you next month. Kids these days... ■

Real Markets: Watching Trends in the Audio Acceleration Market

This is about as exciting a time as you could hope for in the audio acceleration market. "The PC audio chip market is expected to top $700 million this year and grow to over $1 billion by the year 2000. This massive buildup is due to the need for advanced audio features for DVD movies, cutting-edge games with special sound effects, and a whole raft of new technologies that are becoming available to mainstream PC and add-in board manufacturers," said Geoff Ballew, senior industry analyst at Dataquest.

The disappearance of the ISA bus; the emergence of three distinct, but complementary, audio APIs from Microsoft; and lower-cost, higher-performance audio engines are all going to have an influence on the audio subsystem. International Data Corporation (IDC), a leading market research firm, goes so far as to predict that, one way or another, the audio add-in business as it's known today will all but disappear soon after the turn of the century in the United States, and not long afterward internationally.

The first dramatic changeover will occur as ISA boards disappear and are replaced by PCI cards that use ISA hardware for backward compatibility. IDC predicts that PCI controllers that maintain ISA compatibility through software emulation will soon take the lead, but that eventually host-based digital audio will begin to take over. The United States market will drive this evolution, just as it has in other audio trends. Sixty-two percent of all audio in the United States will be host-based by 2001. Worldwide, only 42 percent of all audio solutions will be host-based by 2001, according to IDC.

Mercury Research believes that PCI audio accounted for between 10 and 15 percent of the market, with ISA solutions making up the balance towards the end of 1997. But the company's forecast foresees a shift in 1998 to approximately 25 percent PCI solutions early in the year. By the end of 1998, PCI is expected to account for about a 60 percent share of the audio market (Figure 1).

The big volume in sound card sales overwhelmingly favors Creative Labs, who number among their competitors Aztech, Diamond Multimedia, and Turtle Beach. With over 70 percent of the market share, Creative is going to be hard to displace. That's why, for so many players in the audio market, the transition to PCI audio (and by association, DirectX) appears to be an enor-

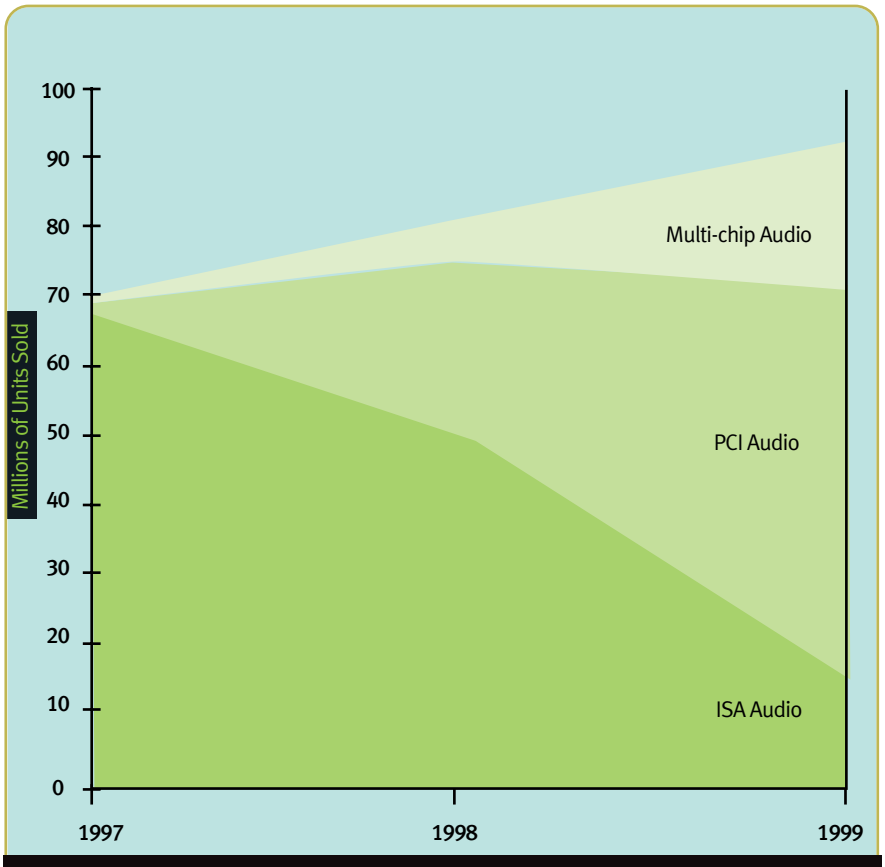


FIGURE 1 *The PC audio market goes PCI.*

Omid Rahmat works for Doodah Marketing as a copywriter, consultant, tea boy, and sole employee. He also writes regularly on the computer graphics and entertainment markets for online and print publications. Contact him at omid@compuserve.com.

mous opportunity. While the legacy of Sound Blaster remains an important part of the history of the evolution of gaming on the PC platform, for game developers this transition offers a chance to redefine audio quality and effects in previously impossible ways. And anything that can enhance the overall PC gaming experience is a good thing.

The Players

So, Creative Labs is the undisputed king of the PC audio market. However, Creative was slow to move beyond the world of its own Sound Blaster ISA-moribund standard to the emerging PCI market. So, Creative used its deep pockets to buy one of the leading PCI audio companies, Ensoniq, and followed this move with the acquisition of speaker-maker Cambridge Soundworks. By signing a couple of checks, Creative has bought all of the components of an audio subsystem, from software algorithms to the chips that drive them to the speakers that make the noise. Creative also has a subsidiary, E-mu Systems, which makes audio accelerators. The company has been quick to move from the higher-cost audio products that it was used to selling to lower-cost products that can find their way into users' hands through PC and systems OEMs.

In its efforts to fight the tide of PCI audio, Creative took its battles to higher ground when it recently announced that the U.S. Patent and Trademark Office had granted it a patent on PCI wavetable audio technology (patent number 5,698,803). Creative calls it Digital Sampling Instrument Employing Cache Memory, and according to the company, this technology permits the audio chip to more effectively utilize system memory. It also covers methods enabling efficient use of the PCI bus on personal computers — allowing the audio chip to handle complex wavetable support without significant additional memory dedicated only to audio, and without requiring time-consuming re-accessing of system memory. The patent represents one in a family of patents rooted in E-mu's wavetable technology.

The first thing Creative did with its patent was file a lawsuit against Diamond Multimedia and ESS Technology. The suit relates to PCI audio technology embodied in Diamond's Sonic Impact family of sound cards and ESS's Maestro-2 chip. And of course, Diamond and ESS are vigorously defending themselves. In the rapidly changing audio market, what's at stake is ownership of audio acceleration on the PCI bus. Diamond's initial foray into audio was based on its Monster Sound brand product, a board that doesn't have legacy support, but is aimed at the game enthusiast who wants wavetable synthesis, positional audio, and all the other goodies that come with high-end audio on the PC.

Will Strauss, president of market research firm Forward Concepts, put things in perspective by saying, "From a chip standpoint, Yamaha and ESS Technologies have dominated the sound synthesis market in the past, but PCI audio has brought in a host of new players, including Cirrus Logic (through their Crystal Semiconductor division), Aureal Semiconductor, EuPhonics/Analog Devices, and even Creative Labs (through their E-mu and Ensoniq subsidiaries)."

Yamaha, once the largest PC audio chip supplier, is now one of the top five suppliers. The company's single-chip solutions offer a good mixture of features and performance. The OPL3-SA has a Sound Blaster compatible interface for legacy systems. The famous OPL3 FM synthesizer that made Yamaha's name is also an integrated codec (compression-decompression device).

ESS Technology owns a third of the PC audio chip market. The company isn't focused purely on audio products, but has built quite a reputation and market position for itself around the Maestro line of PCI audio accelerators. ESS took its market leadership position by integrating FM synthesis in a single-chip solution in the mid-1990s, and although the company had to fight Yamaha over patent issues, the company has grown to be the supplier of one of the largest groups of OEMs, including IBM, Sony, and Compaq.

Cirrus Logic has a wide range of audio products and is among the top three PC audio vendors. Cirrus, like Analog Devices and many other audio chip vendors, is looking at the future integration of audio and modem circuitry. The company also has its hooks in the notebook market, where Cirrus has enjoyed past successes as a graphics chip vendor.

Aureal Semiconductor originally spun off of the infamous Media Vision in 1995. The company not only builds 3D audio solutions, but also licenses its technology to Analog Devices, ATI, Cirrus Logic, Diamond Multimedia, and S3. The company's first PCI controller, the Vortex, combines Sound Blaster compatibility, wavetable synthesis, DirectSound and DirectSound 3D acceleration, as well as support for Aureal's proprietary A3D positional 3D solution.

EuPhonics is unique in the influence it has on the PC audio market because it doesn't make PC audio chips itself. Instead, the company provides the software that supports DSP-based (digital signal processing) audio products, as well as drivers and other core technologies for wavetable synthesis, 3D spatial and localized audio, and Sound Blaster compatibility.

Analog Devices was one of the main proponents and developers of the AC-97 standard for audio. Insofar as Analog Devices has been a pioneer, the company has failed to build a leading business position. Still, the company is

Some other players in the PC audio market.

Company	Audio Expertise
Harris Semiconductor	AC-97 codec.
Motorola	DSPs for high-end audio cards.
Oak Technology	Audio/modem chipset, and audio logic chipsets.
Opti	Variety of solutions. Company has a large presence in market.
Qsound	3D audio software technology.
Rockwell	Wavetable synthesis.
S3	PCI audio solution, SonicVibes.
SAL	Spatializer 3D audio technology.
SRS Labs	3D audio technology.
VLSI	New entrant in the PCI audio market.

a noted player in the DSP chip market. Using its DSP expertise to bring low-cost, but powerful DSP audio solutions to the PC market using EuPhonics software technology, Analog Devices has positioned itself very well for the future. The company's DSPs are found in Diamond's Monster Sound audio accelerators, and it will be interesting to see how it integrates modem and audio functions on future products.

Microsoft has probably the most dominant position in the PC audio market as a result of the control they exert over the operating system. In the Windows 98 WDM (Windows Driver Model), the intention is to remove the need for Sound Blaster compatibility hardware. This will obviously impact the chipsets being built in the future (provided things go according to plan). Of course, Intel is also desperate to increase the features of CPU-centric audio acceleration. This may have an impact on the market, but Intel's desire to eat up cycles sometimes ignores the practical realities of the way software, particularly game software, works. So, whether the host is capable of processing all of the features of DirectMusic, DirectSound, and DirectSound 3D at the same time is debatable.

Trends

For the true game enthusiast, and for most consumers, hardware acceleration of audio is going to remain a key ingredient in the PC gaming experience. There are also a number of technical reasons for this.

Dr. Jeffrey Barish, president and CEO of EuPhonics says, "Concurrency is key, and that is why audio acceleration should be a significant consideration for game developers. With DirectMusic, you have 64 voices, and you have about 32 streams of audio for DirectSound, and finally, about 8 streams for 3D positional sound. The general MIDI standard was 24 voices, and it was up to about 32 voices until recently. So, you can see that the need exists to balance the audio load between the CPU and an audio accelerator to get the most of out of what today's audio standards allow."

Todd Moore, audio product line marketing manager for Diamond Multimedia sees another dynamic in the market, "With extreme price pressure in the sub $1,000 PC market, OEMs and system integrators are being forced to use less-than-perfect audio solutions to remain price competitive. Meanwhile, the most exciting PC audio technology is only available via audio add-in cards."

DSPs have grown cheaper and more powerful in recent years, and seem destined to become even more price/performance competitive in the coming years. Some audio vendors see this as an opportunity to consider future DSP generations as supporting game audio, the general AC3 standard, and maybe even modem support. It may be difficult for vendors to create compelling audio/modem solutions because of the different natures of the two industries, but the advantages of a lower-cost, small-footprint solution are seen as positive enhancements for those companies producing DSP solutions.

And, of course, there is the question of intellectual property rights and who gets to own audio technology. Creative has acquired and developed a very strong portfolio. In the world of 3D positional audio, DirectSound 3D has not proven to be a panacea, and so there are likely to be a number of competing algorithms in the marketplace. DLS, the downloadable sound standard from the MMA (MIDI Manufacturers Association), is a great opportunity for game developers to customize audio palettes and to build more effective audio special effects into their titles. The consumer is getting the best of all these changes. With Windows 98's WDM, we will probably say good-bye to Sound Blaster and move into a new realm of audio development. So, the audio market looks as though it's experiencing a major upheaval for the better.

Yet, one problem remains — the speakers and audio set-ups on the average desktop. Because of the way our ears work, there are fundamental limitations in 3D positional audio. That means the effect with two speakers is very limited. Whether users will treat PC audio solutions the way they treat home theater audio or music stereo is still open to debate. Speakers on today's PCs are much better than they were even a year ago, but the more pricing pressure is put on the PC market, the more add-on components suffer. So, even your greatest audio efforts may end up lost in a haze of distorted speaker sound. ■

front line AWARDS

THE FIRST ANNUAL

BY ALEX DUNNE

PHOTOGRAPHY BY CARTER DOW

If one thing's for certain about game development, it's that it never stands still. Technology and tastes change, and as they do, the tools that usher games along from ideas to gold masters must follow. When we relaunched this magazine over a year ago, one of our editorial goals was to peek into the toolboxes of game developers and honor the best solutions inside. We created the Front Line Awards to do just that.

These awards, which play off the tag line of the magazine ("On the front line of game innovation"), recognize the most innovative and productive software products available for creating professional games. Some of the products you'll see in the following pages were created specifically for the purpose of game development, while others are more generalized tools that happen to work particularly well to solve problems associated with game development. We also decided to honor some consumer hardware products — we felt it would be a great change of pace for game developers to pat the backs of the hardware technology companies that make games look and sound as good as intended.

The Front Line Awards were presented in a ceremony early last month at the CGDC in Long Beach. Because of the lead times associated with magazine publishing, as I write this, this ceremony has not yet happened, and consequently there are no thrilling shots to show you in this issue of people gushing tears as they accept our awards. You can, however, check out these pictures at our web site.

The Categories

The spectrum of products we recognize with these awards is necessarily wide; the many different disciplines required to create games makes this so. There are 13 categories this year, encompassing programming, art/animation, music/audio, production, and the aforementioned graphics and audio hardware. Next ye\ar, it will likely grow to include even more — we didn't want to bite off more than we could chew our first year.

In order to be eligible for this year's awards, products had to have been released within 18 months prior to December 31, 1997. That window of eligibility corresponds to the generally accepted notion that games take about that long to develop; therefore game development tools released during that period likely had the biggest impact in the industry prior to our cut-off date. Each of the following 13 pages showcases the winning products in each category, accompanied by a brief description by one of the judges as to why it was selected.

The Judging Process

If you ask any of the judges listed below, they'll tell you that the judging process was hard work. It began last fall when we began to accept product nominations on our web site from magazine readers and the judges themselves. These nominations made up an initial ballot, which the judges narrowed down to five finalists, then eventually to a winner in each category (receiving the Front Line Award) and two runner-ups (beneficiaries of the Preferred Product Award). In addition, we inducted some venerable products into our Hall of Fame.

To qualify for the Hall of Fame, products have to be at least five years old, and must have had significant impact on game development. Hall of Fame Awards have been handed out to a number of products that no longer exist, but it's comforting to know that some tried-and-true tools have stood the test of time, improving themselves with each new version.

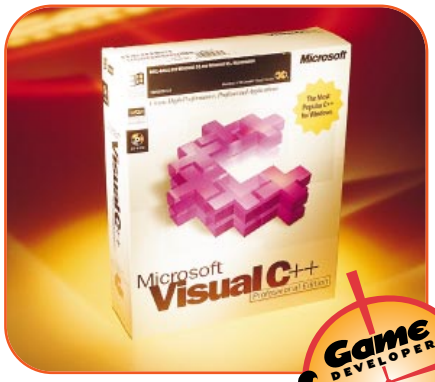
The Judging Cadre

Due to the fact that the magazine wanted to recognize a broad spectrum of products, the judges themselves had to have experience in many different areas of game development. We drafted our columnists, advisory board members, magazine contributors, and professional game developers from our readership for the task, and each person sat on one to three categories that best matched their skill set. The judges this year were as follows:
Dave Bagget • Seamus Blackley • Chuck Carr • Helen Cho • Jeanne Collins • Scott Corley • Tzvi Freeman • Chris Hecker • Brian Hook • Rob Hubbard • Rob Huebner • Mike Kelleghan • Andre Lamothe • Jeff Lander • Mark Miller • Casey Muratori • Larry O'Brien • Bobby Prince • Bob Provencher • Greg Pyros • Matt Saettler • Todd Siechen • Dave Sieks • Bryan Stout • Dan Teven • Dave Thielen • Josh White

It's hard to express the gratitude that the magazine owes each of these people. Their input during the judging process was insightful, educational, and oftentimes funny. Special thanks also go out to the magazine's own Wesley Hall, who helped marshall the judging process this year and kept the Front Line Awards on track.

You may or may not not agree with some of the choices on the following pages. Just remember that all awards, by nature, are subjective. Our judges feel that these products exhibit superior features and/or performance, offer distinct advantages over their competitors, and in no small part have helped move the game development industry forward.

At the end of the day, what more can you ask?



Game Developer front line AWARD 97

PROGRAMMING ENVIRONMENTS

VISUAL C++ 5.0 (MICROSOFT)

In a world full of hype one compiler has always hit the mark — Microsoft Visual C++. Historically groundbreaking in Windows development and implementing the latest technologies, VC++ 5 is without a doubt the best C/C++ compiler for the PC. Version 5 has support for just about every new and old Microsoft technology such as standard Windows SDK programming, MFC libraries, ActiveX, COM, RTTI (Run Time Type Information), and the latest STL (Standard Template Library). In addition, the development studio is one of the slickest in the business, allowing incredible control over compilation, optimization, multiple targets, and more.

For those of you that might be a little skeptical, I tested VC++ 5 against Borland, Watcom, GNU, and more. Although these other compilers, most notably Borland and Watcom, come in a very close second, they simply don't have the tight integration and features that VC++ does. The only downfall of VC++ 5 is in the area of hobbyist — the days of DOS are dead as far as Microsoft is concerned, hence there is absolutely no support for DOS programming. However, this shouldn't be that much of a surprise since VC++ hasn't supported DOS since version 2.0. On the other hand, there is "Console App" support making quick and dirty experimentation and DOS text-like apps still feasible (with 32-bit memory addressing!). And of course, Microsoft compilers have never been known for being resource friendly, so I suggest at least 200MB of disk space and at least a Pentium 100mhz to make compiling a bearable process. Remember, VC++ is a highly optimizing compiler and takes a little longer to perform the code analysis, but the project dependencies are very smart and the compiler supports pre-compiled headers, to speed the process up as much as possible. So, if you want to do some serious Windows or game programming with DirectX then VC++ 5 is the way to go!

—André Lamothe

HALL OF FAME

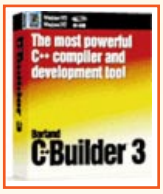
WATCOM C/C++ (POWERSOFT):
This product is one of the most important tools in gaming history. It's the compiler that was used to write DOOM from id Software. Not only is the compiler a "to the metal" product that creates the most optimized C/C++ code I've ever seen for the DOS platform, it targets more operating systems than any other product. Once it became known that Watcom C/C++ created DOOM, developers got a copy and started creating games like wildfire with it and its bundled DOS Extender by Rational Systems, DOS4GW (which also deserves note as the kernel that made protected mode compilers possible).

—André Lamothe

BORLAND C++ (BORLAND):
Borland has almost always been ahead of the technology curve. They had the first integrated graphics, the Borland Graphics Interface (BGI),which allowed you to write graphics games with a single tool. Back when compilers came on a dozen floppies and consisted of a slew of separate editors, compilers, linkers and debuggers, Borland's IDE products fit onto just one. They don't fit on a single floppy any more, but their commitment to quality and innovation remains unabated. Over the years, no one has done it better.

—Mike Kelleghan

PREFERRED PRODUCTS



C++ BUILDER (BORLAND):
C++ Builder provides true visual, component-based development. Just plop a few controls on a form, edit their properties in the object inspector, fill in a few event handlers, compile, and run. This is the way visual development should work.

But is it good for game development? In a word: yes. Whether I use it as my primary development tool, or as a RAD platform for creating in-house utilities, editors and the like, C++ Builder is my compiler of choice. Its well thought-out Visual Component Library makes creating GUI applications a snap. Builder also includes support for COM, ActiveX controls and the Active Template Library (ATL), so programmers aren't limited to what's available in the VCL. Additionally, with support for just about every standard that's important to C++ programmers, Win32, MFC, STL and finally COFF object files (note to DirectX programmers: no more IMPLIB!), Builder can do everything the other guys can do, and more. *— Bob Provencher*

CATALYST/TORCH (NEWFIRE):
Catalyst was very different from the IDEs that dominate this category. Not only was it intended for a narrower audience (developers of real-time 3D games), but it was intended for a narrower part of the development cycle. It could create VRML 2.0 worlds, or you could take assets that you'd already created plus Java or C++ code that you'd already written, hook them together, and tweak.

Catalyst's UI had a few rough edges, but the rendering engine, Torch, was fast, feature-rich, and deployable either as a DLL or a browser plug-in. Torch could use a variety of rasterizers and made good use of hardware acceleration. It also had facilities to analyze and improve the playback speed of a scene.

Unfortunately, Newfire ceased business operations after the Front Line Awards judging took place. This sad news underscores the risks inherent in building tools for game developers. We hope to see more products as ambitious as Catalyst next year.

— Dan Teven

API designers inevitably have to balance the two opposing goals of abstraction vs. efficiency. At one end of the spectrum, an API does little good if it requires programmers to deal with hardware minutiae to get decent performance. Likewise, no programmer wants an elegant API for a system that's 100 times slower than code written "down to the metal."

OpenGL's architects (and now the OpenGL Architectural Review Board) met the challenge and delivered a 2D and 3D graphics API that performs well across the gamut of graphics hardware, yet presents a straightforward procedural interface that makes sense for videogames, high-end modeling software, and everything in between. OpenGL offers the programmer a sensible idiom: a `glBegin(X)` call prepares OpenGL to receive coordinates for a primitive of type X; subsequent `glVertex()` commands specify the vertex positions; then a `glEnd()` call tells OpenGL to close out the primitive. Building upon this methodology, OpenGL provides a uniform interface for drawing points, lines, triangles, quads, n-gons, triangle strips and fans, and even NURBS surfaces. Programmers use the same syntax to draw to a display list or remote machine.

OpenGL's `glBegin ... glEnd` abstraction nicely shields the developer from hardware details, but maps onto typical rendering hardware capabilities well enough that the performance loss is tolerable. Furthermore, since its inception, OpenGL has dictated hardware design — Silicon Graphics' rendering hardware deliberately targets OpenGL, and now many 3D accelerator card manufacturers do too. This trend should continue to lessen OpenGL's performance penalty for video games. Interestingly, however, one of OpenGL's unsung virtues is that it isn't meant just for games. This means that game programmers can use the same API for (and even share code between) a run-time game engine meant for a Pentium, and design tools destined for the team's R10000 or Alpha workstations.

Direct3D vs. OpenGL politics aside, this API is well-designed and feature-packed, and able to meet game programmers' demands in both run-time and tool code. *—Dave Baggett*

PROGRAMMING LIBRARIES AND APIS

OPENGL 1.1 (OPENGL ARB)

HALL OF FAME

DIRECTX 5.0 (MICROSOFT):

Naming DirectX 5 a preferred product is a bit like naming rain a preferred weather condition. It's sometimes inconvenient, and often messy,

but we couldn't get along very well without it.

DirectX 5 boasted improvements in almost every area, but the force-feedback API was one of the most popular new features. And the addition of `DrawPrimitive` to Direct3D took some of the urgency out of the D3D/OpenGL debate.

Previous DirectX releases legitimized Windows as a game platform by tunneling through several layers of operating system overhead, by providing standard interfaces for hardware acceleration, by guaranteeing a lowest common denominator through hardware emulation, and by taking the burden of device support away from applications. DirectX 5 continued these trends by giving display hardware access to DirectDraw surfaces, by opening up DirectSound3D to hardware acceleration, by adding an API for audio input, and by solving compatibility problems caused by upgrading device drivers. *—Dan Teven*

GLIDE (3DFX):

3D accelerator cards went from expensive developer tools to affordable consumer playthings when low cost cards finally hit the market. The outstanding cards in the batch were all based on the same chipset, the 3Dfx Voodoo Graphics. For developers eager to take advantage of this card, 3Dfx provided Glide, a no-nonsense API that's easy to learn and provides the functionality a developer needs to take full advantage of the Voodoo Graphics chipset.

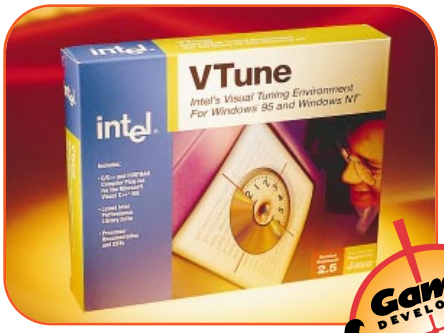
At a time when standard 3D APIs were unsupported or in development, 3Dfx provided a way for developers to build successful games with the Voodoo Graphics chipset. The approach that 3Dfx took with this API was appreciated by many developers — many wish it was the only one required.

The game world needs its common APIs for 3D cards, but for now, supporting 3D cards directly has definite performance advantages. Glide provides a good example of direct support without any hassle.

—Scott Corley

THE MILES SOUND SYSTEM (RAD GAME TOOLS):

Before RAD Game Tools bought it, the Miles Sound System used to be known as AIL. Most people simply called it the "Miles drivers." Regardless, this library was essential for DOS game development because it solved the thorny problem of sound card compatibility. The drivers just plain worked — and John Miles arguably deserves a place in the Hall of Fame for that alone. But the high quality of the product extends well beyond the drivers. MSS features the same clean programming interface under both DOS and Windows, efficient resampling and mixing, low playback latency, and good documentation. John Miles also gave great customer support. It's no coincidence that Microsoft based version one of DirectSound on the Miles mixer. Although MSS needed little improvement, RAD has diligently added new features: Redbook audio, for instance. RAD's also continued the tradition of great support. MSS is nearly as valuable today as it's ever been. *— Dan Teven*



Game Developer front line AWARD 97

PROGRAMMING UTILITIES

VTUNE 2.5 (INTEL)

Back in the good old days, game programmers could safely ignore the few K of code in the ROM charitably called "the operating system" and just code directly to the hardware in assembly language. But not anymore — now developers must write applications that coexist with multitasking operating systems and access hardware only through APIs. Generally speaking, this means using a high-level language (HLL) like C or C++.

High-level languages have numerous advantages — they make most programmers more productive, and HLL code is more maintainable and portable than assembly language. The fact remains, however, that code designed specifically for a particular architecture and hand-written in assembly language can often outperform even carefully crafted HLL code by an order of magnitude.

Fortunately, most programs have only a few performance bottlenecks. Optimize these routines, and the program will run dramatically faster. The obvious compromise, then, is to write the majority of a game in an HLL and a few critical sections in assembly language. The hard part is *identifying* those critical sections.

Enter VTune. Run your program through VTune, and VTune will tell you where the hotspots are. Of course, in practice it's more complicated that that, and making full use of VTune requires time to learn its intricate interface and mastery of Intel x86 assembly language programming.

VTune exemplifies massive overkill. No one writing a standard Windows application needs to drill down to the assembly level and stare blearily at instruction timings and explanations of why two opcodes won't pair on a Pentium.

But game programmers, almost alone among developers these days, often *do* need to do just that. A 3D action game tangibly improves with every cycle saved, and VTune, more than any other profiler on the market, gives us what we need: the sledgehammer of optimization tools.

VTune can analyze programs using time-based sampling, event-based sampling (using a Pentium Pro or Pentium II's built-in sampling hardware), or statically, by estimating the performance or a piece of source code. It even has a "code coach" that recommends source code changes. New to version 2.5 is JIT-compiled Java support.

Intel — you get a gold star. And in software, of all things!

— Dave Baggett

HALL OF FAME

BOUNDSCHECKER (NUMEGA LAB):

What strikes fear into the hearts of game programmers everywhere? The chance that their game engine contains a memory leak or overwrite. Fortunately NuMega's BoundsChecker finds both memory and API errors. BoundsChecker is really two tools in one. It helps catch memory leaks, array boundary violations, and heap overwrites, all without recompiling the code. It also validates all the Windows API calls made by a program, to ensure that the application adheres strictly to the API rules and to avoid problems on current and future Windows platforms. *— Rob Huebner*

SOFTICE (NUMEGA LAB):

Why was SoftICE — a debugger that's normally used on operating systems and device drivers — selected to a game development Hall of Fame? Because a difficult bug in an operating system or a driver has turned up during almost every game development campaign. In those situations, SoftICE is a capable solution an much less expensive than hardware debuggers.

This rock-solid product has significantly improved year after year, even though it dominates the third-party debugger market. *— Dan Teven*

PREFERRED PRODUCTS

MSDN LIBRARY (MICROSOFT):

Back in the early days of Windows programming, programmers had to struggle with somewhat incomplete documentation and virtually no information about bugs and issues that might be important to them. But Microsoft changed all that. MSDN Library is a quarterly compilation of virtually everything they know about their products. Well, OK, there's probably some super-secret information that doesn't make it onto the MSDN Library, but it's no less remarkable for that — it's what does go into the MSDN Library that is astonishing. Documentation on virtually all of Microsoft's SDKs as well as the complete knowledge base (KB) of bugs and information of use to programmers. All of this packaged with an easy-to-use viewer with powerful query capabilities make it a must have for every programmer's library of tools.

— Bob Provencher

SMARTHEAP (MICROQUILL):

SmartHeap provides two invaluable services. First, it has the best heap (that's `new` and `malloc`) debugging checking ever. It finds and fixes heap errors like overwriting the end of the heap, accessing `Free`d memory, and so on. It also provides much faster heap management than Visual C++. In addition to the standard heap allocation routines, it provides more specialized heap allocations.

SmartHeap can create a heap where all allocations are the same size. This makes it blazingly fast and eliminates fragmentation for that heap. It can also create pools where you can allocate multiple times within the pool, and then free the entire pool in a single call (as opposed to thousands of `free` calls if you called `malloc` thousands of times for that pool).

SmartHeap is invisible until it either finds a bug or you want to use one its advanced heap management calls. Your program will speed up because the standard heap calls are so much faster.

— Dave Thielen

Game Developer front line AWARD 97

STOCK MEDIA

6000 SERIES EXTENSION (SOUND IDEAS)

The Sound Ideas 6000 Series Extension is a collection of sound effects. Although it was difficult to compare the divergent products in this category, I think the results reflect current industry practice. Most game developers don't use stock 3D models because they often prefer to create these from scratch. I don't, however, know of a single game sound developer who doesn't regularly use sound effect libraries.

A good sound library allows game sound designers to take advantage of hours of field work done by professionals in an affordable, well organized, convenient format. Sound libraries come in many configurations, some as specialized as to contain only airplanes or automobiles. While these are valuable if you need the exact sound of the door opening on a 1996 Ford Taurus, they are generally too expensive and specialized for widespread adoption. What game developers really need is a great, all-purpose, well recorded, comprehensive collection of sounds at a reasonable price. In 1992, Sound Ideas introduced just such a collection, the 40 CD 6000 series (AKA The General). The General quickly became a cross industry (games, film and TV) standard.

Building on this success and feedback from their users, Sound Ideas decided to put out a sequel to the 6000 series, called the 6000 Series Extension. Over a year and a half was put into gathering survey results from their users, planning, and digitally recording new sounds to complement and complete the original. David Yewdell (*Starship Troopers*, *Fifth Element*) has contributed over two CDs worth of sounds to the Extension. As with the 6000 series, all of the sounds are digitally recorded, clean, and well mastered. According to Sound Ideas, over ten times as much material was recorded as was used, and each sound was quality controlled by at least three people before being inserted in the library.

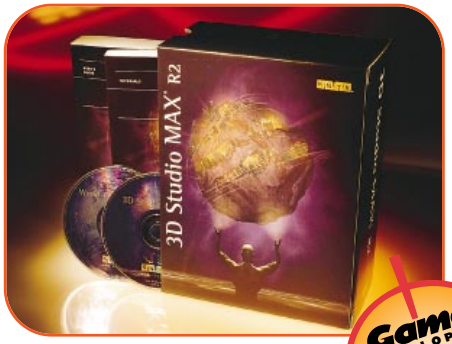
Overall, the 6000 Series Extension admirably fills in many of the holes that existed in the 6000 Series with offerings such as a full complement of switch and button sounds. Ambiences are smooth, loopable, and free from clicks and other distracting artifacts. Impact sounds are meaty. There are nice collections of new doors, crowd sounds, firearms, and telephones. With the addition of the Extension, the 6000 Series from Sound Ideas is now the single most essential piece of stock media for any game development company. — *Mark Miller*

18 Perfect People (Zygote):

Human characters are one of the most difficult things to model in a 3D program. This is where 18 Perfect People comes in handy. This CD-ROM contains high quality 3D meshes of perfect people. In the collection, you will find male, female, and child meshes, both clothed and unclothed. There are two styles of meshes, separate objects for the body parts and single mesh versions. Best of all, the disc contains models in all the popular 3D file formats including 3DS, DXF, OBJ, and HRC. This makes it easy to bring them into your favorite modeler. The polygon counts on these models are a bit high for real-time applications: about 10,000 polygons on average. However, they can be easily reduced or modified to lower polygon counts. The collection will certainly give you a great head start on creating your own 3D characters. — *Jeff Lander*

Concept:FX (F7 Sound and Vision):

I have to admit that when Michael Oster contacted me about his "Concept: FX" CD-ROM, I was hesitant to take a listen. I did listen though, and it was worth the time. This CD is a collection of 195 "unusual" royalty-free sound effects for the PC and Mac in five formats (AIFF @ 44.1khz, 16-bit stereo, Internet AIFF @ 11.025khz, 8-bit Mono, WAV @ 44.1khz, 16-bit stereo, multimedia WAV @ 22.05khz 16-bit stereo and Internet WAV @ 11.025khz 8-bit mono). "Royalty-free" is important: This means that one can use the sounds without having to pay for each use. The sound effects on the CD were recorded with a microphone that mimics the human head. The result is an effect that is "like being there." Those recorded sounds were then heavily processed. This is not your typical sound effects CD. Effect file names range from the somewhat descriptive ("Automatic Door," "Broadcast Interference") to the completely non-descriptive ("Coronary SubString," "X Glub"), so you will want to spend some time reviewing all of the files in order to become familiar with what's available (make sure to take notes). There is plenty of raw material here for further digital editing, mixing, etc. For "Concept: FX 2 – For Professionals Only," I'd like to see more effects, minimal processing and one generic file format. — *Bobby Prince*



Game Developer front line AWARD 97

MODELING AND ANIMATION ENVIRONMENTS

3D STUDIO MAX 2 (KINETIX)

3D Studio MAX is a wonderfully rich environment of 3D tools, sometimes so much so that it becomes rather overwhelming. Game development has relied heavily on 3D Studio since the early days, and MAX has successfully carried over those capabilities to the new environment for 3D content. With the wealth of new features popping up in 3D tools over the past couple of years, MAX has done well with its open architecture and the many third-party plug-ins. The biggest innovation in MAX has been the Modifier Stack in which you can edit and remove multiple modifiers applied to objects. This tool is a major innovation in 3D interface functionality, and one that deserves serious attention.

MAX R2 has done a tremendous job of fixing many of the shortcomings of MAX R1, like wireframe redraw speeds, the material editor, and the addition of the new NURBs toolset. MAX's competitors all seem to fall short in one area or another, some are too expensive, some are lacking in simple interface functionality, and some are just too sparse on tools and features. MAX, however, has a reasonable price for game development, a great toolset, and a very open architecture for added functionality and customization. MAX also has the largest third party plug-in library available for any 3D software out there. It happens to be my favorite tool at the moment, and I plan to use it heavily on a variety of projects including game development, movie special effects, and any other 3D need I may have. MAX has a very comfortable and familiar interface that I come back to when I know I need to do something spectacular. It places the limitations on the artist, and not on the software. I well up just thinking about how great MAX R3 will be, considering how much of an improvement R2 has been over R1.

— Todd Siechen

HALL OF FAME

3D STUDIO 1.0 (KINETIX):

3D Studio accidentally caused a revolution in 3D game artwork. Its ported-DOS-entangled architecture and the weird, frustrating interface and general insensitivity to real-time game developers' needs certainly didn't propel it to stardom. No, its success came from its embodiment of the concept: Power to the people.

Power in 3D Studio meant a complete set of 3D face editing tools: vertex, edge, face, object, keyframe. 3D Studio didn't protect you from turning an edge and making spaghetti out of your geometry, and game artists rejoiced for this power. Like DPaint, 3D Studio semi-accidentally offered artists direct access to technology limits in a visual way. The power was extensible through plug-ins (IPAS).

Less obvious was the "to the people" philosophy. 3D Studio offered most of the power of workstation 3D software at a tiny percentage of the cost. This mass marketing approach led to the revolutionary idea of conveying emotion with a 3D model. Once thought weird and "arty," emotion became a widespread goal. Though it was far from perfect, 3D Studio defined a new class of game development tool, and made a huge difference in game development.

— Josh White

PREFERRED PRODUCTS

POWERANIMATOR 8 (ALIAS|WAVEFRONT):

PowerAnimator 8 has an advanced toolset for trimmed NURBS modeling. Despite the depth of the program, the workflow is very good, possibly the best. You can easily customize the interface by creating resizable tool shelves. Configurable marking menus give quick access to the commands that you use frequently. In comparison, MAX's workflow is often too intricate to function efficiently.

PowerAnimator has two renderers: a raycaster and a raytracer. The two together produce great atmospheric, volumetric, and optical effects.

True volumetric particles in PowerAnimator 8 create smoke, atmospheric effects, sparks, and fire in a variety of realistic ways. Fluids and lava are achieved with blobby particles. Particles generating hair and fur can cast shadows and respond to wind and motion.

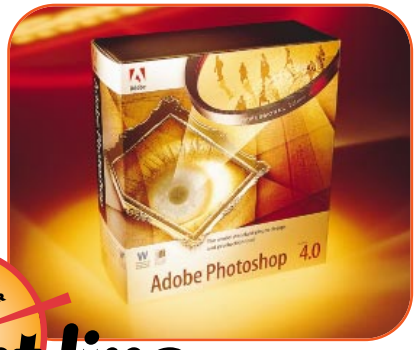
Though PowerAnimator 8 may be slow with polygonal models due to the fact that it's NURBS-based, it's still an excellent choice for modelling and rendering.

— Helen Cho

SOFTIMAGE 3D 3.7 (SOFTIMAGE):

Everyone has seen the amazing work produced by Softimage. This modeling and animation software has been the main tool in creating such hit movies as *Starship Troopers* and *Men in Black*. But Softimage has also become a very strong product for game development. In version 3.7, Softimage really put forth some effort in creating great tools for games. The polygon modeling is very strong. However, the product really shines with its animation. The tools for creating lifelike 3D characters are simply the best out there. The new Rendermap feature in Mental Ray makes creating shadow maps, used in games like QUAKE, a breeze. The Sapphire (SDK) and new Game Developers Kit (GDK) make it easy for developers to create custom plug-ins for extracting information out of the program. You pay for this performance though. Softimage is much more expensive then some other 3D packages. However, when you really need the power to create amazing 3D characters and worlds, Softimage simply can't be beat.

— Jeff Lander



Game Developer front line AWARD 97

IMAGE EDITING AND MANIPULATION

PHOTOSHOP 4.0 (ADOBE)

Heard of it? Photoshop is the de facto standard image editing software in the game development industry. This momentum is not merely the result of going with the flow. Photoshop's broad base also translates into a superabundance of resources for users of all skill levels: third-party books, articles, discussion groups, freely shared techniques, and special effects plug-ins abound for this ubiquitous image editor.

This atmosphere of vitality is matched by Adobe's improvements to the software in Photoshop 4. Not only has the interface been cleaned up for a smarter work environment, it's been standardized throughout other Adobe products like Illustrator and After Effects (Front Line Awards Preferred Product, 2D Animation & Video). This helps make Photoshop part of a well-integrated suite of professional tools that will benefit any game artist.

Compositing layers, a feature first introduced in Photoshop 3, is made even more powerful in this release with the addition of adjustment layers. These allow color and tonal manipulation of multiple image layers without the need to collapse or flatten them. Image tweaking has never been easier.

Another Photoshop 4 improvement of particular interest to the game artist is the new Actions Palette for automating repetitive tasks. Macro-commands for batch processing can be created as easily as pressing a "Record" button. Once recorded, these "actions" can be simply edited in list form without the need for busy artists to also master a scripting syntax.

This latest upgrade is also peppered with worthwhile improvements such as more flexible selection behaviors, broader EPS import capabilities, and much more. The enhancements expand the professional image-editing tool game artists have come to rely on for texture creation, retouching of 3D renders, color indexing, painting backgrounds and much more. — *Dave Sieks*

PAINTER 5 (META CREATIONS):

One of the first lessons professional developers learn is: Use what works. That's why Painter's success against Photoshop is so amazing. The differences in technical capabilities between the two — both are Windows-based 24-bit paint programs — are minimal. Painter offers game artists traditional art tools. Painter seduces artists that love traditional natural media. With no apologies to its digital foundation, it offers a much closer simulation of a real, nibbled-wood, smearing-graphite pencil than anything before or since. It combines paper's tooth, pencil's nature, and other detailed realities of natural media into a digital painting package. Besides simulating traditional media, it also embraces an innovative feature set.

All of this aside, Painter's popularity is strong evidence that game artists don't see their work as sterile chrome-sphere graphics, but as true artwork in a new medium, needing a new breed of artist's tools. — *Josh White*

PHOTO-PAINT 8 (COREL):

In the last five years I've used every paint program in existence including Photoshop, Fractal Painter, Paint Shop Pro, DPaint, and more. All of them have their pros and cons. Photoshop is an incredible image processor, but it lacks good drawing capabilities. PSP is a great program, but lacks layers and advanced color control. But when I got my claws on Corel Photo-Paint, I was truly impressed. Sure it's impossible to be good at everything, but PhotoPaint comes damn close! It has the coolest interface I have ever seen, and I think it blows away Photoshop like it's standing still.

The image processing and lighting filters are amazing, as is the color control. In addition, you can actual draw and paint with Photo-Paint — it's not just an image processor with drawing functions attached at the hip. If you want an all-around high-end paint and image processing program then Photo-Paint 8 is it. — *André Lamothe*

HALL OF FAME — DELUXE PAINT (ELECTRONIC ARTS):

Deluxe Paint (DPaint) is the ideal Hall of Fame winner. Its dominance spanned 10 years, peaking during the 16-bit cartridge boom, and DPaint still pulls plows for a few developers today.

What's so great about it? By accident as much as intent, DPaint reveals the bare wires of computer graphics in a way artists can handle. For example: RGB sliders go to a bit-baring 255 (not an artificial 100%), and they skip every few numbers (2,3,5,6,8...). DPaint shows you what is really being stored when you slide the slider. Try avoiding palette entries 124-128 as you create an 8-bit image in Photoshop, and you'll understand why DPaint's stencils are supposed to be jagged.

However, the real miracle of DPaint was its sheer fun hiding under those ugly icons. Turn on symmetry, hand over the mouse, and you can silence six-year-old visitors (the only true benchmark for pure entertainment). Especially since the frustration of pixel-by-pixel painting has faded from our memories, DPaint is truly legendary, and DPaint in-jokes are surefire conversation starters with any veteran game artists.

— *Josh White*

Game Developer front line AWARD 97

2D ANIMATION & VIDEO SOFTWARE

DIGITAL FUSION (EYEON SOFTWARE)

Digital Fusion 2 from Eyeon Software, Inc. is an extremely powerful compositing and special effects program for the Windows NT (Intel, and Alpha) platform. It takes full advantage of NT's multi-threading and multiprocessor capabilities to support its incredibly fast renderer. Digital Fusion uses a layout system called 'flows' to organize the entire composite and/or effects in a logical and straightforward manner. Each of the numerous tools included with the package has full spline-based controls for smooth adjustments. A built-in tracker makes it easy to composite moving areas, such as replacing the sign on a truck as it is driving down the highway, or add a lens flare or glow to a moving light source. The tracker can also be used to stabilize shaky images.

Eyeon has been successful in attracting some of the best plug-in creators to Digital Fusion. Ultimatte's blue-screen matting technology is not only available as a plug-in, but is included in the higher-end film version of the product. The 5D Monster plug-ins currently consist of six sets of high quality Flame effects plug-ins transferred from their original SGI versions.

Release 2 added sound support to assist in lining up the video, as well as a more traditional timeline view into the project. Drag and drop of clips and effects, as well as the ability to queue up multiple renders were also new.

The best feature of Digital Fusion can be summed up in one word: speed. Speed in setting up flows and especially speed in the rendering process itself. Depending on how much memory you have, you can tweak the program to load in many files in advance, so that everything is ready when it is time to render the frame. This is a very professional, well designed product that just keeps getting better, and deserves to be a part of every game developer's software tool chest — *Greg Pyros*

HALL OF FAME

DeBabelizer (Equilibrium):

DeBabelizer has become the de facto standard for translating, compositing, and batch processing multiple graphics files in the game industry, with support for over 100 different file types. It has handled the graphics processing, palette optimization, and file translation for a large percentage of the games produced in this decade.

Starting out as a Macintosh-only program in 1991, DeBabelizer now has both Mac and Windows 95/NT versions available. Until the Windows version shipped at the end of 1996, some game developers who had already switched to Windows kept a Mac around for the sole purpose of running DeBabelizer.

Palette manipulation has always been one of DeBabelizer's strongest areas. When all games were utilizing a 256-color screen, palette control was one of most time-consuming and thankless aspects of game production. DeBabelizer can set palettes, remap and reduce colors, create custom palettes, sort, and distribute colors as needed. The program has become a classic and well deserves its Hall of Fame status. — *Greg Pyros*

PREFERRED PRODUCTS

After Effects 3 (Adobe):

Game artists will appreciate Adobe After Effects's ability to layer and combine rendered 3D animations,

digital video, and static bitmapped images with precise control, and will also be pleased by its color correction tools.

After Effects is well-integrated with Adobe's other flagship products, Premiere, Illustrator and Photoshop (Front Line Award winner, Image Editing and Manipulation), making it part of a powerful suite of tools for the game artist. Another great feature is Bezier paths for animated masks. These can be made to travel around the screen and even change shape over time. Also extremely handy is a text tool enabling type to be animated along complex paths.

Beyond the standard feature set, many third party plug-ins are available to provide additional functionality, as are books, training videos and online users groups. After Effects may not be the sexiest digital video editor available, but it is an incredibly useful and powerful package for the money. — *Dave Sieks*

Premiere 4.2 (Adobe):

Premiere 4.2 is a solid 2D digital video editing program available on the Windows 95, Windows NT, and the Power Macintosh. It allows film, video, multimedia, and web content creators to combine still images, movies, audio, video, and graphics on their desktops and output to video, multimedia formats, or the Web.

Premiere was one of the first programs with an extensible architecture, allowing many third-party developers to create plug-ins for the product. As this is going to press, Adobe is announcing the 5.0 upgrade for Premiere. Promising an improved user interface, better integration with other Adobe products like Photoshop and Illustrator, more control over transitions, enhanced Edit Decision List (EDL) support, CD-quality audio support, and long-format editing tools, Premiere will be an even stronger contender in the future. — *Greg Pyros*

Game Developer front line AWARD 97

AUDIO RECORDING AND EDITING

SOUND FORGE 4 (SONIC FOUNDRY)

Back in the early '90s, the Mac dominated the audio development landscape. PC hardware was weak. DOS and early versions of Windows were a mess, and PC MIDI and digital audio editing tools were lame or non-existent. Yet all of the audio development systems for PC games themselves ran on the PC. It was under these oppressive conditions that I added the first PC to my studio (a 286, no less) and chalked it up as a necessary evil. While suffering, I came across one truly great product, Sound Forge.

Sound Forge 4 is a truly innovative product, created by a great company (which is staffed and run by very smart, nice, reasonable people), which is rock solid, really easy to use, dominates the market, and rapidly evolves to embrace and incorporate industry standards and practices.

At first, I used Sound Forge just to open alien files that were sent to me by other game developers. Then I discovered that the sample rate conversion was actually better than my Mac editor. Pretty soon, I found that all I was doing on my Mac was digitizing material. I'd then take it over to Sound Forge for editing. Much to my surprise, I was beginning to prefer this upstart Windows product over my tried-and-true Sound Designer II.

Sound Forge has remained squarely positioned at the leading edge of PC audio development. The new version adds or improves on past versions with DirectX audio plug-ins, Internet audio support (Java, RealAudio, and NetShow On-Demand), multi-level undo/redo, AVI file support, crossfade loop functionality, phaser and wah-wah effects, an input vs. output dynamics graph, multi-band dynamics, highly accurate resampling, dithered/noise shaped fades, stereo expansion and MS mixing, and much more. Sonic Foundry offers tools that complement and enhance the value of Sound Forge by creating a comprehensive and integrated working environment. Today, the Windows PC has become the dominant platform in the development of game audio largely on the strength of Sound Forge.

— Mark Miller

RECYCLE 1.6 (STEINBERG):

ReCycle is a cross-platform audio application that creates audio file "slices" of imported drum loops and grooves. By analyzing the waveform peaks, ReCycle creates "slices" of the loop. It will then assign each slice a MIDI note number, create a key map, and transmit everything to a sampler or Cubase VST as sample files and a program file. With my K2000 I use SMIDI to send data back and forth, including the .MID file that ReCycle creates from the original loop. Since the loop's parts are mapped out across the keyboard and you have a .MID file of the groove in your sequencer, changing tempo and creating different drum fills is made easy. If you're unfamiliar with working with drum loops it may not seem that exciting, but if you are, you'll find it quite appealing. It's definitely a new way of working with digital audio. As far as competition goes, there isn't any. At present, ReCycle is the only product of its kind on the market. *— Chuck Carr*

THE NATIVE POWERPACK (WAVES):

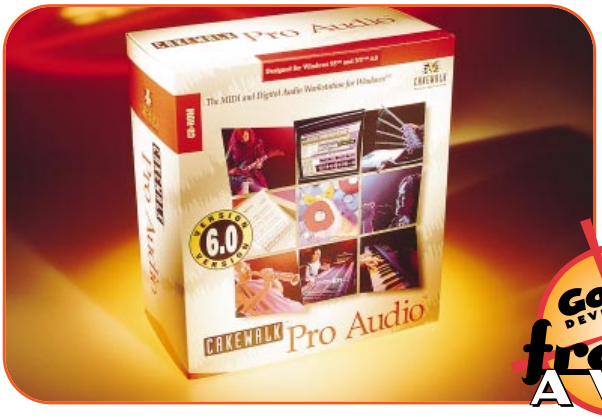
This tool is a set of audio software processors made up of 6 software plug-ins and one application. The plug-ins, available for both PC (DirectX) and Mac platforms, include: C1 (compressor/gate), L1 (an awesome look-ahead peak limiter), S1 (stereo imager), Q10 (paragraphic EQ), TruVerb (reverb/space processor), and IDR (Increased Digital Resolution), a ditherer/noise shaper. My favorite and most used is L1. When you really want to get a sound file with that "in your face" sound without clipping, it's gold. C1's compressor presets are clean and warm. I suggest getting the free update to C1+ at www.waves.com. The speech compressor/downward expander preset in C1+ is ideal for voice dialogue. The price you would pay for the hardware equivalents of these plug-ins makes the cost very appetizing. With more audio applications supporting plug-ins, it's nice to have the Native Power Pack in your virtual rack of gear.

— Chuck Carr

HALL OF FAME

SOUND FORGE 1.0 (SONIC FOUNDRY): When Sound Forge 1.0 was released, it offered exceptional digital audio editing power at a price small game developer could smile about. It created high quality "movie-like" sound effects and included many features found only in high-end dedicated tools at the time. Finally, affordable and powerful digital audio editing was possible on a standard PC. Today Sound Forge still pushes the "digital audio editing envelope" thanks to plug-ins from Sonic Foundry or third-party developers. *— Bobby Prince*

COOL EDIT 1.0 (SYNTRILLIUM SOFTWARE): Cool Edit quietly entered the digital audio editing market as shareware. It didn't seem like much. A closer look revealed that it had the power of some of its retail competitors. This software handled basic digital audio editing well and qualified as a full-fledged sound processor. Highlights include script/batch processing, support for many popular sound file formats and excellent sample/bit rate conversion. Because of its great power at a minimal price, it became the favorite digital audio editor for many small companies. *—Bobby Prince*



Game Developer front line AWARD 97

AUDIO COMPOSITION SOFTWARE

CAKEWALK PRO AUDIO 6 (CAKEWALK)

Considering its popularity and wealth of powerful features, it's no surprise that Cakewalk Pro Audio took the award in its category. It's a 32-bit, 256-track MIDI sequencer/digital audio workstation for Windows. StudioWare controls any MIDI device through software panels. Using moving sliders and knobs known as "widgets," MIDI-controller or system-exclusive data is transmitted and can be recorded to a sequencer track. Veteran sound designer Bobby Prince first introduced this feature to me by creating a StudioWare panel for my Roland Alpha Juno 1. The Juno responds to system-exclusive data in real-time. With Cakewalk's StudioWare, I now have the ability to automate my aged but loved synth. Another strong feature in Cakewalk is the ability to create CAL (Cakewalk Application Language) programs. CAL is "an interpreted language for writing custom editing commands." With CAL, it's possible to create macros of your favorite keystroke commands and save them for easy reuse. More sophisticated users can write CAL routines to perform tasks not found in Cakewalk, such as creating chords from single notes, thinning data from tracks, and creating arpeggios. This offers unlimited possibilities in automating tedious and sometimes complicated tasks.

If you use real-time effects, you'll love Cakewalk's support of DirectX plug-ins. Effects can be inserted into each digital track and chained together, allowing all the tweaking and sweetening you wish. While composing a song for SHOOTOUT 98, I had 16 tracks of digital audio playing with real-time effects (reverb, compression, chorus, and so on) inserted on each track using a Pentium Pro 200MHz with 80 MB of RAM. With its plentiful third-party support and continuous effort to upgrade its technology, Cakewalk Pro Audio continues to be a favored MIDI/digital audio workstation among game developers. *—Chuck Carr*

HALL OF FAME

CAKEWALK PRO FOR WINDOWS 1.0 (CAKEWALK):

DOS MIDI sequencing was a compatibility nightmare. In 1991, Windows 3.0 arrived, creating a standard way for MIDI software to speak to sound cards. Anticipating this, Cakewalk launched the first truly Windows-compatible sequencer, Cakewalk Pro for Windows 1.0. This tool quickly became a standard for PC game audio developers.

Many things set this product apart from its competitors. Foremost was its strict compliance with Windows conventions. Cakewalk was the only PC sequencer to include integrated notation editing. Finally, Cakewalk allowed users integrated access to Windows' MCI commands. This allowed a crude, but workable method for synchronizing MIDI composition to digital audio files on a single CPU.

As testimony to its HoF status, Cakewalk has racked up a serious list of top game credits that include everything from DOOM to REN & STIMPY'S QUEST FOR THE SHAVEN YAK.

—Mark Miller

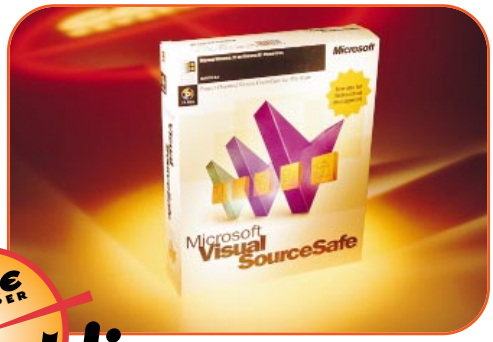
PREFERRED PRODUCTS

LOGIC AUDIO 3 (EMAGIC):

This digital audio sequencer works as effectively as a comprehensive package as it does as a "simple sequencer." It's software that allows the user to push the envelope for home and project studios. Its extensive feature set (which includes sequencing, machine control, mix automation, notation and digital audio recording) is flexible, and almost every feature can be customized. For those who don't let sophistication stand in the way of the possibilities, there's almost unlimited musical production power here. One highlight of this software is "Screensets," which allow you to save up to 99 window configurations (each Screenset can include multiple copies of an editor window, each with different parameters). This can save many hours of having to size, position, re-size and re-position windows. With software such as this, there's no excuse for not working on your (next) hit right now. *—Bobby Prince*

STUDIO VISION PRO 4 (OPCODE):

Opcode has always struck the right balance between innovation and standardization, and between power and ease-of-use with this product. The tool combines the latest incarnation of Vision with some great bundled add-ons to offer a fully featured, well-integrated suite of tools for composing and editing synchronized MIDI and digital audio. You can edit MIDI and digital audio in a synchronized and uniform environment that supports all of the standard sequencing conventions. What sets this product apart is the superior integration with Digidesign's Protools hardware (it supports increased audio I/O, full TDM bussing, and Samplecell TDM), their excellent digital audio pitch shifting and time compression/expansion technology, the bundled patch librarian (Galaxy 2.1), and the other CD goodies (Waves EZ Verb, and Arboretum Systems Hyperprism.) *—Mark Miller*



Game Developer front line AWARD 97

PRODUCTION SOFTWARE

VISUAL SOURCESAFE (MICROSOFT)

Since almost every professional game development team uses source control, and the vast majority of them are using SourceSafe, it seemed a natural nominee for these first annual awards. But does SourceSafe deliver on its promise of project-oriented source control? Is it appropriate for a game programming environment? This year's judging panel voiced a nearly unanimous "yes."

SourceSafe began life as an third-party tool developed by One Tree Software. In 1994, Microsoft bought One Tree and the product became Visual SourceSafe. Newer versions of SourceSafe included tighter integration into the Developer Studio suite and additional features useful to web developers. Throughout the product's history, it's managed to stay one step ahead of its competition in terms of power and ease-of-use. SourceSafe offered a GUI-based interface before many of its competitors, and offered easier tools for administration and setup.

SourceSafe offers the ability to store and track revisions on binary files as well as source code, making it a viable tool for asset management. Its ability to merge different branched versions of a file with a minimum of fuss continues to improve. SourceSafe is adept at working with multiple projects, and has extensive features for tagging, pinning, and sharing files across projects.

SourceSafe is not completely without warts. Its database can sometimes become corrupted or suffer from file locking problems. Most of these problems are easily fixed using simple repair tools, and the mean time between these failures is high. Performance can be sluggish at times. Its high network bandwidth use also makes it cumbersome to use over dial-up connections. Previous update versions have suffered from performance bugs, but Microsoft made quick repairs.

Source control software is an indispensable tool for most game development teams, and SourceSafe is the best tool for the job. If Microsoft avoids the temptation to overload the product with web-centric features at the expense of the traditional programmer, Source Safe is positioned to continue its reign as the source control solution of choice for game programmers.

— *Robert Huebner*

IMAGEBLASTER 2.2 (KEYLABS):

Many of today's game designs involve changes to the Windows 95 (and soon Windows 98) operating system. Specifically, if a game needs DirectX to run, that installation changes the operating system. Since developers can't count on users already having this component installed and configured correctly, they need to test and re-test their installation code. But how do they quickly get the machine back in the pristine non-DirectX environment without re-formatting the hard drive and reinstalling everything from scratch? How do they test multiple operating system environments without investing in several machines? They can make an image of the machine's boot drive using Imageblaster before running their installer on it then blast it back to the saved image when they're through.

KeyLabs

Imageblaster has built-in file compression and fault tolerance. Once you've created a disk image, you can use it over and over again. These images are simply files which can be catalogued, compressed, and stored on-line for future use. This means, given enough storage space, you can have an image for all the various operating systems you need to test, and virtually expand your testing lab's capabilities tenfold. ImageBlaster is fully file-system independent. It can handle any PC file system including all FAT file systems, HPFS, NTFS, and NetWare, to name a few.

— *Jeanne Collins*

MSDN LIBRARY SUBSCRIPTION (MICROSOFT):

You can choose not to use Microsoft compilers. You can choose not to use Microsoft Office. You can choose not to use DirectX. But you cannot do without a subscription to Microsoft Developer Network (MSDN). From its humble beginnings as a CD-ROM when players were so scarce Microsoft bundled a discount offer for a single-speed player the size of a pizzabox to its current multiheaded incarnations as CD/website/bookshelf-binder-full-of-everything-Microsoftian, MSDN is the default source for technical information. The level of writing is surprisingly high (aside from the annoying jokes of Dr. GUI) and a great deal of the technical information is unavailable elsewhere.

— *Larry O'Brien*

Game Developer front line AWARD 97

CONSUMER GRAPHICS HARDWARE

VELOCITY 128 (STB SYSTEMS)

Based on the RIVA 128 chipset, the STB Velocity 128 is an outstanding 2D/3D video card solution. It's still early in the history of consumer-level 3D graphics hardware, and for performance reasons, many people are willing to put up with 3D-only solutions. But the STB Velocity 128 proves that you don't have to give up performance or an extra PCI slot. This card performs well all around.

The Velocity 128 brings some serious fill rate to the table, giving game developers some breathing room. The 3D feature set is fairly complete, although as a developer you'll find a few things missing if you're used to the 3Dfx — and the RIVA chipset drivers do some funky things with their automated MIP-map generation. But for a circa-1997 single card 2D/3D solution, the 3D kicks butt. For 3D to become a huge market, 2D and 3D have to work in perfect harmony, and the Velocity 128 proves that it can be done. The pack-ins with this card show off the speed, and the performance via Direct3D is up there as well. STB seems to be committed to providing solid OpenGL drivers, though like many OpenGL drivers these days, they are constantly under development.

So what's it got on the 2D side? The 230Mhz RAMDAC made me happy. High speed and good VESA support should make everybody happy. Compare that to some other 2D/3D cards from 1997 that had crummy RAMDACs and really crummy VESA support. The 4MB 100Mhz SGRAM kicks out some nice resolutions, like 1600×1200×65k or 1024×768×16.7M. These resolutions are easily enough for home use, and they come with higher than average refresh rates to boot.

The 3D card market has a long way to go, and as with most other PC technologies, what we think is great this year will be collecting dust next year. But for 1997, STB moved the industry one step closer to excellent 2D and excellent 3D in every machine.

— Scott Corley

PREFERRED PRODUCTS

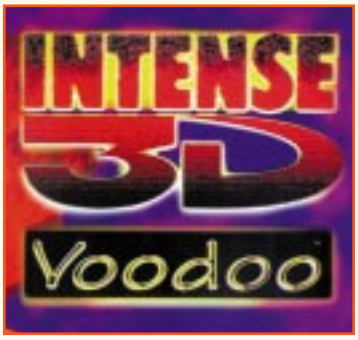
MONSTER 3D (DIAMOND MULTIMEDIA):

The 3Dfx Voodoo Graphics chipset changed the face of game development on the PC forever, but it would have been a classic case of "the best doesn't always win" if it weren't for the support of some top notch video card companies. At the start of the 3D accelerator craze, Diamond was already at the top of their game delivering high quality video cards, and the Diamond Monster 3D was one of the first 3Dfx-based cards that everybody felt comfortable purchasing.

The onslaught of high quality 3D hardware at an affordable price from a well known brand name took 3D acceleration from speculation to certainty. The success of this full-screen-only card that doesn't support anything but games has shown the world the market pull of gaming and the importance of 3D to gamers. While many other companies came out with 3Dfx-based cards, the Diamond Monster was the one that got the buzz and the shelf space. Will consumers grow up and demand more than full-screen-only 3D this year, leaving cards like the Monster and its successor behind? 8-ball says: not likely.

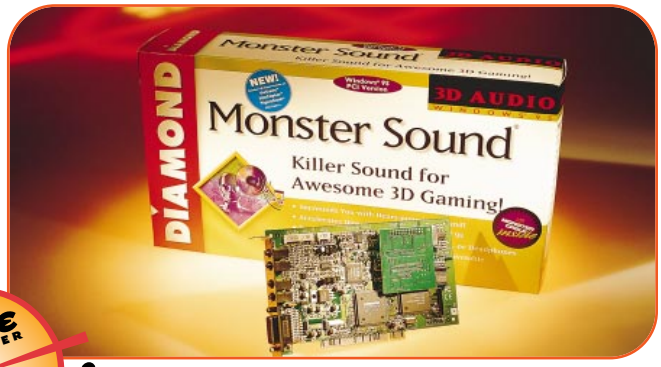
— Scott Corley

INTENSE 3D VOODOO (INTERGRAPH):



3D Graphics hardware is finally living up to the hype. Of the cards I tested, three made it to the finals: the Intergraph Intense 3D Voodoo, the STB Systems Velocity 128, and the Diamond Multimedia Monster 3D. The Intergraph Intense Voodoo based on the 3Dfx chipset is just amazing. The card is both a 2D and 3D solution, plus it has an additional video out signal for television output, which makes it a true gaming video card. The card features all of the bells and whistles that make games look good such as perspective correct texture mapping, level of detail, anti-aliasing, bilinear, and trilinear texture interpolation, and a whopping 6MB of VRAM. Sure, other cards may have some or all of the same features, but the bottom line is that in this card they all work. I without hesitation recommend the Intense Voodoo to anyone that wants a sub $200 full 2D/3D solution with state-of-the-art 3D performance.

— André Lamothe



Game Developer front line AWARD 97

CONSUMER AUDIO HARDWARE

MONSTER SOUND (DIAMOND MULTIMEDIA)

Out of all the products nominated for a Front Line Award, the Diamond Monster Sound was one of the few that truly broke new ground. As delivery media and system memory increase in capacity, and DLS becomes a prevalent standard, games will devote more and more space to sound effects and instrument samples. Typical high-end games will require fast, efficient sound mixing that's not limited by fixed memory on an audio card or gated by slow download times. The Diamond Monster Sound meets these requirements with solid scatter-gather audio streaming over the PCI bus. This allows a game to keep its large sound effects and sample banks in main memory, but frees the CPU from performing mixing and other DSP operations. Throw in decent 3D sound spacialization, support for a second set of speakers, and a standard extension port for upgradable wavetable synthesis, and you're looking at a great little audio card. It'll even pass-through your old SoundBlaster or similar legacy card in case you still have DOS games that won't work with the Monster's SoundBlaster emulation mode.

The card was not without its shortcomings, however. The card's MIDI support is not very solid, and its wavetable sample set is of poor quality. Worse, on some systems and cards, the included MIDI module daughtercard does not work at all. Fortunately, it is replaceable, and the card cooperates nicely with higher-quality add-on daughtercard (such as the DBXG50, another one of the products nominated for this award). Hopefully, as games begin to use DLS for their music, the Monster's drivers will begin using the card's native capabilities to play the samples supplied by DLS instead of those on the daughtercard, eliminating the card's primary shortcomings. MIDI troubles notwithstanding, the Diamond Monster Sound is a big step in the right direction for consumer audio cards. Hopefully, the new cards of 1998 will all follow its lead.

— Casey Muratori

AUDIOTRIX 3D-XG (MEDIATRIX):

The Audiotrix 3D-XG is a PC soundcard that features the Yamaha XG sound set for high-quality MIDI music, and full duplex 16-bit digital audio. It also features an on-board DSP that can be used to process digital sample data or MIDI.

Musicians and gamers will notice a big improvement over the sound quality of older FM-based cards, and cards that use much lower quality wavetable sound sets. The 3D-XG responds to MIDI controller messages for reverb and chorus, as well as the more exotic XG features such as filters and other XG MIDI controllers. Games that use extensive MIDI sound tracks will roar with this card.

The board supports full duplex audio — great for the future Internet-based games that use speech, and also for games that use speech recognition.

— Rob Hubbard

SOUNDBLASTER AWE64 GOLD (CREATIVE LABS):

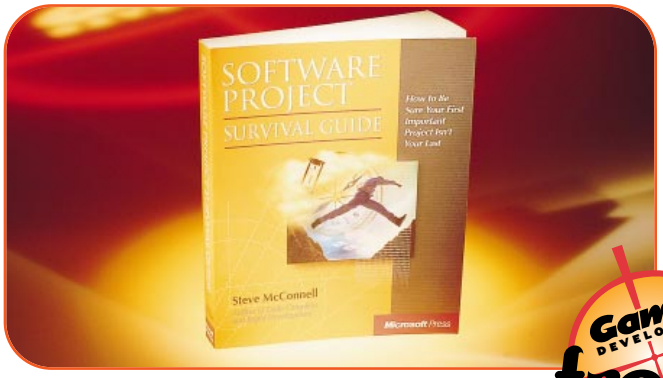
This card features 4MB RAM for wave table sound sets, full duplex audio, and EMU's SoundFont technology that enables users and games to download their own sounds. Users can download a 4MB general MIDI sound set for high quality MIDI music, and musicians can utilize a further 32 voices using the Waveguide software synth. The card is superior for aspiring musicians who want to get into PC music creation. It even supports a SP/DIF digital audio output for connection to professional music equipment. It's also a good upgrade for old SoundBlaster-16 users who are having performance problems with DirectSound. Many games now have dedicated SoundFont support for sound effects or music. One minor problem — don't set the MDI device to Waveguide and expect to run a game. Finally, the card comes with lots of software goodies at a great price.

— Rob Hubbard

HALL OF FAME

SOUNDBLASTER 1.0 (CREATIVE LABS): In the beginning of PC sound cards, there was the AdLib. It was a Yamaha FM synthesizer chip that allowed software products to include a music soundtrack. Innovative game designers of the time could program the FM synth to make basic sound effects, but it was not possible to have life-like sounds in a game. This all changed with the release of the SoundBlaster 1.0. By including a digital sound chip in addition to the FM synthesizer chip, the SoundBlaster revolutionized sound for the PC computer gaming industry. For the first time, it was possible to have life-like and believable sound effects in a game. For quite some time after its initial release, the Sound Blaster was the only PC sound card with these possibilities – a situation that not only cemented it into the minds of many millions of users, but also made the word "SoundBlaster" synonymous with "computer sound card."

— Bobby Prince



Game developers are always looking for the silver bullet that will fix their development problems. Long delays, failed projects, low quality software, long hours, and bad morale still run rampant in the game industry, and developers are so shy of silver bullets that many discard new ways of thinking outright. There is no one way to make game development easier, so the familiarity of status quo plus some wishful thinking keeps many projects in the "at risk" category.

Here's your new silver bullet. The difference here is that the *Software Project Survival Guide* (SPSG) isn't a single silver bullet, it's more of a silver shotgun blast.

Game Developer front line AWARD 97

BOOKS
SOFTWARE PROJECT SURVIVAL GUIDE
BY STEVE MCCONNELL (MS PRESS, 1997)

And it isn't as easy as just pulling a trigger. There are a ton of things that you have to do before the SPSG considers your project "Outstanding… virtually guaranteed to succeed in all respects, meeting its schedule, budget, quality, and other targets." Sound hard? Sound like an impossible goal? At around 260 pages, this book reads fast. If you're die-hard skeptical about any fancy new-age approach to software development, this book is all over you. The only prerequisite for loving this book is: you must have been on at least one hellishly difficult project in your life. If you meet that prerequisite, this book will speak to you. Do you hate unnecessary paperwork and superfluous processes? SPSG is written with you in mind. You will recognize every suggestion in this book as something that, at the end of some project, you wished you had done from the start.

This book may not have an impact on the way many games are made, and for every project that succeeds because of SPSG, there will be another smash hit that was created by disorganized, seat-of-the-pants, garage development. But I'd be willing to bet that in every case, the SPSG team got more sleep. — *Scott Corley*

HALL OF FAME

THE MYTHICAL MAN-MONTH
BY FREDERICK P. BROOKS, JR.
(ADDISON-WESLEY, 1975)

When this book came out, many software projects were coded in assembly, memory was scarce, and new and changing hardware was the norm. This is often the situation today in game development. Brooks draws not only on his own experience from working on IBM mainframes, but also uses data and observations from other projects of the time. Readers will find many of the items are still valid today. — *Matt Saettler*

COMPUTER GRAPHICS: PRINCIPLES AND PRACTICE
BY FOLEY, VAN DAM, ET. AL.
(ADDISON WESLEY, 1990)

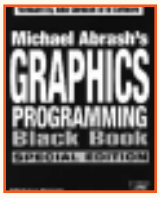
Those who post the question "what one 3D book do I have to own" to USENET seem to always get back the same answer: "Foley and van Dam."

This book covers a huge spectrum of 3D topics in a concise, understandable style. The book transcends operating systems and programming languages to provide the 3D graphics basics in a way that will always be relevant. Implant this book into any brain that is beginning a career in 3D graphics.
— *Scott Corley*

PREFERRED PRODUCTS

MICHAEL ABRASH'S PROGRAMMING BLACK BOOK
(CORIOLIS BOOKS, 1997)

Abrash is a legend in game development. Any book he writes receives a lot of attention, and rightly so. Based on content (solid, real information from people who know), relevance to professional game developers, and writing quality (easy to access, clear, well written, good table of contents and index), Abrash's book scores well. Despite its technical nature, nonprogrammers can learn a lot by reading it — even the parts that are about 80386 optimization (in other words, totally outdated) give insight into how programmers think, which is very valuable for nonprogrammers. Article-compilation books often lack organization, and compare poorly to books that guide a reader through the points in a premeditated, organized way. In this case, however, it's very convenient to have this body of Abrash's writing in one place. — *Josh White*



THE ULTIMATE GAME DEVELOPER'S SOURCEBOOK
BY BEN SAWYER
(CORIOLIS BOOKS, 1996)

If you're in the game industry, you should already know everything in The Ultimate Game Developer's Sourcebook (UGDS). If you're in the game industry and you don't know everything in UGDS, you should brush up on the areas that you don't know. If you're not in the industry, if you read it you might be able to convince girls that you make games (whether this will help you achieve your goals, I can't say). This book attempts to include every bit of information you might need to be a game developer, including game design, content creation, software development, business issues, and legal issues. Get past the out-of-date bits, and the fact that this 5 pound book contains 1.5 pounds of information, and see if there's something in there you haven't seen yet. — *Scott Corley*

Incorporating Motion Capture Animation into an AI Engine

by Wyeth Ridgway

In the game industry today, developers are much too eager to toss motion capture into a game design just because it's currently hip to do so. However, before going through all the work of adding motion capture animation to your game, make sure that motion capture is the best solution to your problems. This article will provide programmers who are familiar with 3D programming techniques with the information needed to get motion capture animation data into a game engine and work with it effectively. Throughout this article, I refer to the Viper game engine and to techniques used in the development of the first game released using Viper, SPECOPS: RANGERS LEAD THE WAY. [See this month's Postmortem on page 74 for details about the game. -Ed.] It would be useful to download and run through the game's demo, which is available at www.zombie.com/games/index.html

To determine whether motion capture is a viable solution for your title, take my Motion Capture Preparedness Test in Table 1. Regardless of whether or not you pass, you'll find this article useful. Why? Because animation data, regardless of its source, still needs to be integrated into the game AI.

Creating the Data

We used Alias's PowerAnimator 7.5 (running on SGI High Impact Extremes) to handle the character motion for SPECOPS. PowerAnimator is one of the best packages available for working with animations, and it allowed our artists to clean up animation sequences very quickly. Late in the project, when we needed a few animations that had been accidentally omitted from our motion capture list, PowerAnimator was powerful enough to create hand-crafted animations that were of sufficiently high quality that they did not stick out when compared to the motion capture animations.

We chose BioVision as our motion capture studio. BioVision has a good reputation in the industry, was affordable, and provided us with animation data that slid right into PowerAnimator. BioVision provided our artists with an 18-bone hierarchical skeleton, which we then scaled and built our geometry upon (Figure 1). We chose not to use any geometry for the shoulder bones in order to simplify the polygonal complexity, but the shoul-

Wyeth Ridgway was the lead programmer of SPECOPS and the technical director of Zombie. He is currently on sabbatical in his hometown of Tucson, Ariz., where he's working on the next great thing. He can be contacted at WSR@Primenet.com.

ders still needed to be present so that the biceps could have a reasonable rotational pivot.

Another constraint we faced was that there couldn't be any animation translation data associated with any of the bones — only the parent of the entire hierarchy (the hips) could have an active translation channel. Because of this, our render loop was able to run faster because it had less data to work with, which is important because hierarchical models are extremely time consuming to draw. With all of this in mind, we had BioVision provide us with an animation file that fit our needs, so we could get working on the support code.

Getting Motion Capture Data into a Game Engine

After the artists applied a basic motion (walking) to our geometry within PowerAnimator, it was time to reproduce that same animation within the game. PowerAnimator has a handy, if only slightly supported export format called .RTG (real-time games format), which contains both the geometry and the animation data. The contents of .RTG files can be displayed in text format, making the files very easy to view and debug. In fact, I wrote a parser for both the geometry and animation files without any format spec (although I'm sure one is available) — to me, this is the sign of a great file format.

Once the hierarchical geometry files are loading and being drawn correctly inside an engine, it's time to tackle the more difficult problem of applying animation to the joints. It's fairly easy to hack together something that runs through a single animation in a loop — in fact this is a great first step in verifying that your math is working correctly. However, managing the overlaps between a hundred animations running in response to user input or AI can be a far more difficult organizational problem.

The first step is to identify everything that your animated character(s) needs to be able to do in the game environment. Walking, running, jumping, and firing a weapon are certainly a common subset of this list. Once you and your game designers have brainstormed the required moves, you need to organize this list into a format that the game can process.

This is where hero positions become useful (see "Animation Terms" for a definition of "hero position" and other terms). To describe how animations will transition into and out of one another, we can define a set of hero positions from which the character may enter various animations. From any given hero

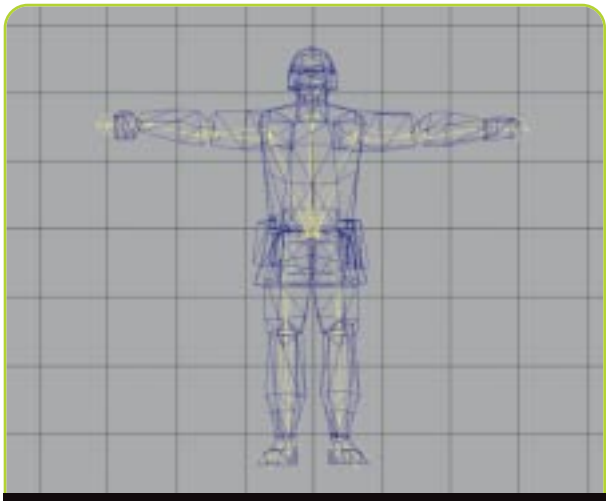


FIGURE 1. *BioVision's 18-bone hierarchical skeleton.*

Animation Terms

Animation Sequence Listing: An AI state and the animations that need to be played to get into and out of that state.
Base Animation: The animation that all animation sequences share at their root. Also, the animation of the primary hero position.
Finite State Machine: A process that is described in states and their relationships.
Hand Animation: Animation data generated by an animation tool, such as 3D Studio MAX or Softimage 3D
Hero Position: A rest position from which animations can be played.
Motion Capture: Animation data captured from a physical source, such as transmitters attached to a person.
Primary Hero Position: The basic hero position from which all animation sequences are described.
Secondary Hero Position: A hero position that is described in terms of its relation to the primary hero position.
Transition Sequence: The sequence of animations that are used to move a character between any two states.

TABLE 1. *Motion Capture Preparedness Test.*

Are the majority of your game's animated creatures going to be motion captured?	If not, motion capture might not be the best solution, because it tends to make hand-animated creatures look robotic and crude.
Is there extensive character motion, such as rolls or gymnastics, that would be tedious or impossible to reproduce by hand?	If not, hand animating the motion should be strongly considered, as today's tools do it well. While motion capture is an excellent resource for realistic character movement, forgoing motion capture will save you money by not forcing you to deal with some of its related problems.
Can the motions of your characters be adequately motion captured?	A good deal of the motions you need might be too difficult to motion capture. If your hero needs to bend, jump, or move in an inhuman manner, motion capture might not be an option. Once you hash out the entire motion tree for your characters, send it to the motion capture studio so they can tell you which animations will cause problems.

FIGURE 2. CROUCH_HERO's entry sequence.

position, you can list out which animations are valid to use. Table 2 lists some of the hero positions and animations from SPECOPS. We needed three basic hero positions: standing, crouching, and lying prone.

For simplicity, you should keep the number of hero positions fairly low. It can be confusing to work with more than five or six, and the transitions between them become more time consuming. If you have animations that seem to require their own hero positions, or that don't slide right into your list, you should strongly consider removing or rethinking them.

From this list of all of your game's animations, you can create an animation sequence list. This is the primary set of information that your game's engine uses at run time to figure out how to transition between animations. The sequence list associates a desired state that the character wants to enter with a sequence of animations that it can play to get there.

The Viper engine uses a text file for the animation sequence list, which can be easily modified by the artists and game designers. Table 3 shows the entries for SPECOPS' standing hero position, along with a couple of animations that are played from that hero position.

EntrySequence and **ExitSequence** are keywords that denote how to get into and out of the state that we're describing. The next token is the name of the state that we're describing — for instance **STAND_HERO**. Finally, we have a list of any number of animation files, which need to be played in sequence to arrive at the desired state. Again in Table 3, you can see that to fire the weapon from the standing position, the character must first play the STANDHERO.ANI animation and then the STANDFIRE.ANI animation.

Note that the first animation file of an **EntrySequence** is always the same as the final animation file played by the corresponding **ExitSequence** statement (see the emphasis in Table 3). This is the primary hero position's animation. All transitions must either start or end with this animation. The animation engine uses the primary hero position to figure out how to transition between animations. I will explain this in more detail shortly.

In Table 4, you'll see the **RegisterCallback** keyword and its arguments. We associate callbacks with certain animation frames to cause game events at specific points in a sequence. For example, grenades need to leave characters' hands during the right frame, and bullets must fire at a specific time. Looking back at Table 4, we see that we want the character to receive a **CALLBACK_STAND_FIRE** message on frame 10 of the STANDFIRE.ANI animation. This message could be a function pointer, a method name, or many other things depending on how you manage objects in your game.

Once the data from these text files has been loaded into your animation engine and its data structures, you're ready to begin playing back animations and bringing life to your character. Let's look at some simple ways to use the information that we now have loaded.

TABLE 2. SPECOPS' hero positions and associated animations.

Hero Position	Associated Animations
Stand	
	Stand and Fire Weapon
	Stand-walk
	Stand-run
Crouch	
	Crouch and Fire Weapon
	Crouch-walk
	Crouch-run
Lay Prone	
	Lay Prone and Fire Weapon
	Prone-crawl

Working with the Data

To get a character to perform some basic actions, you need to maintain a few additional pieces of information. During run time, you can store the animations that you need to play in an animation queue. You should also keep track of the current state that the character is in, as well as the state that it will be in after the animation queue is empty.

For example, assume that the character starts at rest, that the animation queue is empty, and that the current state and target state are both **STAND_HERO**. Since the animation queue is empty and the target state is equal to the current state, the animation engine has nothing to do, so the character will remain at rest in its hero position.

Now assume that the player presses the Fire button (or the AI sends a "fire weapon" message), causing the target state to change to **STAND_FIRE**. A transition sequence needs to be added to the animation queue so that the character will start to animate into the new state.

Anytime we're starting a new animation, the animation engine will execute a simple algorithm. It will push the current state's exit sequence onto the animation queue and then append the target state's entry sequence onto the queue. Let's use this algorithm in our example.

First, the exit sequence animation STANDHERO.ANI, which is used to leave the current state, is put in the animation queue. Next, the entry sequence for the **STAND_FIRE** state is added to the queue.

Note that in SpecOps, all exit sequences end with the character in the primary hero pose, and all entry sequences begin in the hero pose. So we know that the last animation in **STAND_HERO**'s exit sequence will be the same as the first animation in **STAND_FIRE**'s entry sequence. As a result, we don't need the first animation of the entry sequence (it would cause the animation to stall momentarily as two identical frames with the character in the hero pose were played). Instead, we append the rest of **STAND_FIRE**'s entry sequence into the queue. This assures a perfectly smooth transition between animations.

With the target state now set to **STAND_FIRE**, we are finished setting up the transition. The animation engine will start playing the animations in the queue (removing them from the queue as they're completed), and the engine will eventually get to a callback that was set up on frame 10 of the STANDFIRE.ANI. This is the point at which the AI can actually do the mathematics to fire the bullet. The animation engine will then finish playing the entry sequence.

Whenever an entry sequence is completed, the animation engine executes a second algorithm. It pushes the target state's exit sequence into the animation queue and then appends the current state's entry sequence. Finally, it sets the target state equal to the current state. Let's apply this to our example.

First, the animation engine adds the exit sequence for the target state, which is **STAND_FIRE**. Next, the engine adds the entry sequence for the current state, which is **STAND_HERO**. Because the **STAND_HERO** entry sequence has only STANDHERO.ANI in it, and this is already the last element on the queue, the engine does not add the animation to the queue again. As before, to transition correctly, this animation shouldn't be added twice. Finally, the engine sets the target state equal to **STAND_HERO**, and the engine is finished setting up the transition.

The animation engine continues to play the animations in the queue, removing them as they finish playing. When it completes the last frame of STANDHERO.ANI, the current state is equal to the target state, and the queue is empty; the animation engines does nothing more.

Notice that for nonlooped animations, the current state is never set equal to the target state. The character is only in the target state for a single instant before the exit sequence is appended and the character returns to its previous hero position. In some instances, we want the character to remain in the state with which the animation is associated, such as when we're entering a new hero position or when we're playing a looped animation.

Entering a New Hero Position

In SpecOps, when we want to transition from one hero position to another, we don't automatically append an exit sequence onto the queue when an entry sequence has completed. Instead, we set the current state to the target state and leave the animation queue empty. In effect, this leaves the character on the last frame of whatever animation sequence was last in the queue, and it gives us the functionality we need to move between hero positions. In the Viper engine, animation sequences that result in the character posing in a new hero position are denoted in the text file with **HERO** as a suffix, such as **STAND_HERO** in Table 3.

Playing a Looped Animation

In SpecOps, looped animation files are denoted by the word "loop" in their names, such as WALKLOOP.ANI (in Table 3). After the entry sequence is complete, if we want an animation to play continuously until interrupted, we use the **LOOP** keyword, which sets

TABLE 3. **EntrySequence** *and* **ExitSequence** *entries for SpecOps' standing hero position.*

EntrySequence	STAND_HERO	"STANDHERO.ANI" HERO
ExitSequence	STAND_HERO	"STANDHERO.ANI"
EntrySequence	STAND_FIRE	"STANDHERO.ANI" "STANDFIRE.ANI"
ExitSequence	STAND_FIRE	"STANDHERO.ANI"
EntrySequence	STAND_WALK	"STANDHERO.ANI" "TOWALK.ANI" "WALKLOOP.ANI" LOOP
ExitSequence	STAND_WALK	"WALKLOOP.ANI" "WALKEND.ANI" "STANDHERO.ANI"

TABLE 4. *The* **RegisterCallback** *keyword and its arguments*

RegisterCallback	"STANDFIRE.ANI"	10	CALLBACK_STAND_FIRE

the current state equal to the target state until the target state is changed either by user input or an AI message. At that point, a new transition sequence is pushed into the end of the animation queue.

We've now seen how the character can take a simple nonlooped action, transition to a new hero position, and perform a simple looped animation. To see the real power behind these mechanics, we need to look at a more involved solution.

A More Complex Example

Let's look at what's involved in animating the character into a crouching position and getting it to fire its weapon. First, we need to add support for a secondary hero position. Table 5 shows the animation sequence for a secondary hero position called CROUCH_HERO. As you can see, the secondary hero position has an animation sequence that describes its relation to the primary hero position. We've also appended the sequence with the keyword HERO so that the animation engine will handle the transition correctly.

Generally speaking, the player would generate an I/O event representing his or her desire to transition the character from stand to crouch. In response, CROUCH_HERO would become the target state. As described previously, when the entry sequence for a hero position is complete, we leave the animation queue empty and set the target state equal to the current state, which, in this case, is CROUCH_HERO. Everything progresses as usual, except that at the end of the entry sequence, the character is left in the crouch hero position, because the exit sequence would never be placed on the queue.

Now suppose we want to fire from this position. The player hits the fire button, sending an I/O message to the character. The animation engine places the exit sequence for CROUCH_HERO in the queue and sets the target state to CROUCH_FIRE. The engine then finds STANDHERO.ANI in the queue (the last entry), and adds the entry sequence for CROUCH_FIRE after that (without duplicating STANDHERO.ANI). Let's look at the contents of the animation queue:

CROUCHHERO.ANI
FROMCROUCH.ANI
STANDHERO.ANI
TOCROUCH.ANI
CROUCHHERO.ANI
CROUCHFIRE.ANI.

You can see that the character will stand up, crouch back down, and then fire. While this would look visually correct, it would appear more natural for the character to remain crouched the entire time. We need to modify the algorithm so that it finds the shortest sequence of animations that still transition correctly. In this case, we would like to put only CROUCHHERO.ANI and CROUCHFIRE.ANI into the animation queue. To do this, we need to modify the way that we add the exit sequence. Instead of adding the entire exit sequence at once, we must add the animations one at a time. After adding each element, we'll search for it in the entry sequence that we're about to add. If we find it, we'll add the tail end of the entry sequence into the queue. Then we're done. We'll always find STANDHERO.ANI as the last element of the exit sequence if no other elements are found.

So, in terms of our example, we'll first add CROUCHHERO.ANI to the animation queue. When we search for it in the CROUCH_FIRE entry sequence, we find that it's the third animation. Because we found it, we won't add the rest of the exit sequence. Instead, the engine should add the tail end of the CROUCH_FIRE animation, skipping CROUCHHERO.ANI because it's already there. As we desired, this will result in the following animation queue:

CROUCHHERO.ANI
CROUCHFIRE.ANI

Now, we've managed to enter CROUCH_FIRE just fine, but we still need to make certain that we exit this sequence and return to CROUCH_HERO as expected. Using the new algorithm, we'll add the CROUCH_FIRE exit sequence animations one animation at a time, looking for the animation in the CROUCH_HERO entry sequence. The resulting animation queue will only contain CROUCHHERO.ANI, as we would expect.

With this new algorithm for solving for transition sequences, this animation method can handle just about anything. You should be able to sketch out the steps to transition from the middle of a walk entry sequence to a crouching fire sequence and see that it picks the optimal path of animations. Combined with the robust system of animation callbacks, this method represents the majority of the animation logic used in SpecOps. It should just as easily handle any animation requirements, regardless of the source of the data or characteristics of the skeleton.

TABLE 5. *The animation sequence for secondary hero position* CROUCH_HERO.

EntrySequence	CROUCH_HERO	"STANDHERO.ANI" "TOCROUCH.ANI" "CROUCHHERO.ANI" HERO
ExitSequence	CROUCH_HERO	"CROUCHHERO.ANI" "FROMCROUCH.ANI" "STANDHERO.ANI"
EntrySequence	CROUCH_FIRE	"STANDHERO.ANI" "TOCROUCH.ANI" "CROUCHHERO.ANI" "CROUCHFIRE.ANI"
ExitSequence	CROUCH_FIRE	"FROMCROUCH.ANI" "STANDHERO.ANI"

Wrap Up

This system describes most, but not all, of the animation logic used in SpecOps. Some elements have been simplified for clarity, but this overview should provide an excellent starting point for working with your own system of animation logic. Still, a few additional notes should be touched upon before you get started.

INTERPOLATION. I chose not to support interpolation of motion between animations. Interpolation is a valuable tool in making a character seem more responsive. It can allow an animation to be terminated at any point, and the motion into the new animation can be smoothly blended so that it looks visually correct. However, I found that interpolation posed many problems, and I feel that it's useful in only a small

number of real-world cases. It should be considered the final polish to a solid animation system. Refer to Jeff Lander's column ("Better 3D: The Writing Is on the Wall") in the February 1998 issue of *Game Developer* as a starting point for building interpolation into an animation engine.

COMPRESSION. Data compression is a necessity for any animation system. Without compression, you can only store a small number of animations in system memory. I could devote an entire article to this subject alone, but to point you in the right direction, SPECOPS stores Euler yaw-pitch-roll (YPR) values and creates a transformation matrix on demand. This takes longer during the graphics update, but it allowed us to load all the animations for the game at the start of a level.

INTERRUPTING ANIMATIONS. Often it becomes important to halt whatever animation is playing and start something new immediately. This is called an interrupt animation. SPECOPS required this in a few places, where waiting for a transition sequence to complete just didn't look right. For instance, having an enemy stand up so he could fall over dead from a bullet looks strange. Support for interrupts is pretty straightforward and should be part of any animation system.

PHYSICS. Sometimes it's important to be able to animate a character using the physics engine and not animation data. After many attempts to get gunfire to look correct in SPECOPS, we eventually resorted to applying visual noise effects to the gun and chest when the weapon was fired and not using any animation at all. This allows the character to fire in almost any position at any time. You can also get your character to look at a specific object by adding offset values to the Euler YPR before creating the transformation matrix. While this isn't useful in many applications, you should consider supporting it. Realize, however, that it can be time-consuming to add.

GAME LOGIC. Finally, let me make a point with regards to AI structure. I've alluded to the fact that SPECOPS uses finite state machines (FSMs) in conjunction with the game AI. This is a subject to which whole books can be devoted, let alone another article. It has been my continued experience that using FSMs is easier and faster to develop than any other AI method. Better yet, FSMs are the easiest systems to modify once you have a game up and running. It also parallels the object-oriented paradigm, which means it organizes itself wonderfully into the code. I strongly recommend building your game logic, whatever it may be, from FSMs.

Hopefully this article will provide you with everything needed to get a complex animation system into your 3D game. I've found 3D animation to be one of the most complicated but rewarding aspects of 3D game creation. When the character finally starts moving in that true-to-life manner that only motion capture data can create, you really do feel like you've brought something special to life. ■

Acknowledgements

Special thanks to Mark Kreitler for proofing this article, for helping me implement my mad plans, and for being my right-hand man for the last two years.

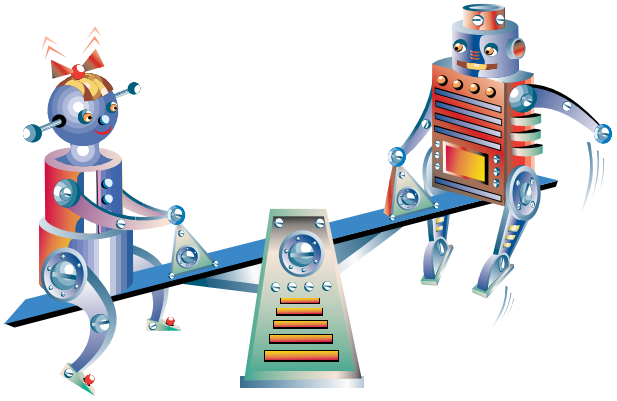
Is Your Game Compatible with Your Users?

by Eric Adams

After spending 18 arduous months creating the ultimate PC game, you immediately take that special someone on a two-week romp in the Bahamas. Well-tanned and refreshed, you enthusiastically return to your company to reap the accolades from your work. However, greeting you is not the director of sales with a big smile and a cigar, but the familiar and somewhat worried faces of the QA and the technical support managers. It seems the game is having problems running on a very popular 3D accelerator card, it doesn't fully install DirectX 5 on certain consumer systems, and the hyped force-feedback effects don't work for users with a certain force-feedback joysticks. How could these problems have happened? You tested the game on the office systems and it worked fine. What went wrong?

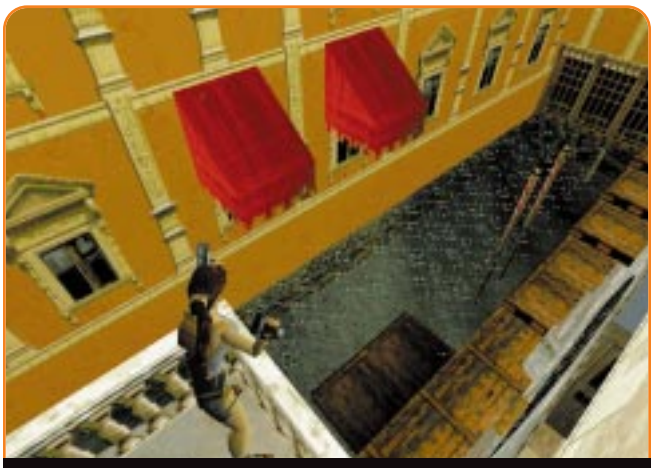
Sadly, hardware incompatibility is common in PC games. The luxuries of console hardware standards don't apply to the world of PC development. Developers and publishers of PC games are burdened by a lack of definable hardware and software standards. Add to this the explosion of new hardware products in a market with thousands of possible system configurations, and you have a compatibility black hole waiting to suck you in.

Compatibility testing is an arduous task that requires commitment and determination by all development parties. The rewards of compatible and stable games may not always be paramount in the minds of hurried developers and publishers. However, hardware bugs diminish the reputation of not only the game, but of the companies involved with it. Bad press, poor reviews, negative word of mouth, and so on cost you sales (and royalties) and make it harder for your next project to attract player interest. Game players have been known to forgive compatibility problems in a mega-hit title (given the quick availability of the ever-needed patch update). However, if you don't have a "killer" title, game players will crucify you and your company for every quality mistake. A marginal game that's saddled with bugs and hardware incompatibilities



Eric Adams is an associate producer and lead tester at Eidos Interactive. He has several years of industry experience at companies such as SSI, BMG Interactive, Domark, and Psygnosis. When he is not gibing opponents along with his QUAKE II Clan (Killer Klowns), he's fanatically following the San Jose Clash. He credits his industry longevity to daily doses of strong espresso and the affection of his terrific girlfriend Janet. Contact him at eadams@eidos.com.

The compatibility testing of JOINT STRIKE FIGHTER focused on high-end hardware due to the technical sophistication of the flight simulator market.



The broad market appeal of TOMB RAIDER II required that the game be compatible with a wide range of consumer hardware.

is quickly relegated to the discount bin. A marginal game that is stable and performs as promised on a majority of systems has a much better chance of gaining an audience.

Tales from the Front Line

As an associate producer and lead tester at Eidos Interactive, I have the unique opportunity to work on both the creative and technical sides of development. In striving to bring users the latest in gaming technology, I also see the downside of pushing the PC hardware performance ceiling. Games with those spectacular graphics at intense speed are ripe for compatibility problems.

In five years of working in this industry, I have yet to see a PC game released that didn't exhibit some type of hardware or software conflict. I've worked on games that have run the spectrum of quality. I've managed games with fifty hours of compatibly testing and endured those with none.

I recently worked on two technologically advanced (and challenging) titles: JOINT STRIKE FIGHTER and TOMB RAIDER II. Both received a moderate amount of hardware compatibility testing. Their stories drive home the sad fact that without a global PC standard, total compatibility is an elusive goal.

JOINT STRIKE FIGHTER. JOINT STRIKE FIGHTER (JSF) is a combat flight simulation that uses an advanced software engine to render the game world and its principals in stunning detail and with excellent speed. Early in this project, I knew that we had a winner. Not only was the game extremely stable and bug free, but Innerloop, the game's developer, was dedicated to making this title a hit.

Our testing resources were limited by two other projects that Eidos was developing in-house at the same time. I was the associate producer and also the lead tester for the JSF project. I had the part-time assistance of a couple of internal and external testers. Mike Mchale (the QA manager) and our producers also lent a hand in multiplayer testing. We were able to test the game on over fifty system configurations, achieving very positive results.

During testing, we focused on 3Dfx-based 3D accelerators, force-feedback joysticks, high-end joysticks (such as Saitek's X36 Control Stick), and high-end 3D video cards (such as nVidia's RIVA 128). The reasoning behind our choice of testing hardware was that the flight simulation market was very savvy in the selection of components. Flight simulator fans researched and bought only the best items. Additionally, JSF was specifically designed to take advantage of these high-end peripherals.

When the game shipped, our only nagging concern was with JSF's multiplayer features. Basically, we had done very little testing on the modem and serial portions of the game. Adding to our concern was the fact that the packet size transmitted between host and client was quite high. This led to TCP games plagued by lag and slow frame rates.

To prevent any confusion on the proper setup and operation of the game, I wrote a thirteen-page readme file on every technical aspect of JSF — from DirectX 5 installation to lists of URLs for major hardware vendors. I was confident that we had taken the steps to circumvent major user problems and decrease the chances of the game needing a patch revision. Was I ever wrong.

After JSF had been on the market for about two weeks, I was dismayed to find a number of newsgroup posts and technical support e-mails complaining of mysterious crashes with the 3Dfx Voodoo Rush cards, a lack of centering force on Microsoft's force-feedback joystick, and a DirectX 5 incompatibility with older 2D accelerator cards. The perplexing aspect of these problems was that we had checked these hardware peripherals; in fact, we'd used them in development. Why and how did these problems only occur with our users?

In the case of the Voodoo Rush crash, we (and the hardware vendors) never found a cause. Fortunately, we found a workaround that worked for most users. The Microsoft force-feedback problem was solved in short order with the kind assistance of the Microsoft Gaming Hardware team. The DirectX issues were resolved by having users update their hardware drivers directly from the manufacturer with WHQL-compliant versions.

One of the areas that we missed in our compatibility tests was the DirectX 5 compatibility of 2D video cards. Because the game supported 3Dfx-only acceleration, we focused on the drivers

for the 3Dfx cards. When the game shipped, we had users report DirectDraw crashes with their 2D cards. These calls accounted for 40 percent of our technical support call volume.

Another lesson we learned (thanks to the force-feedback and Voodoo Rush problems) was that you must try multiple configurations and then follow-up on subtle clues. For example, when we tested Microsoft's force-feedback joystick, we only used two systems and the 2.0 Gaming Devices software. I found that the centering force was poor (the same problem that our users experienced), but I attributed this to the limitations of the joystick itself. I'd read several reviews of this joystick, and poor centering force was a common complaint. Unfortunately, our particular problem was due to incorrect programming with the SDK. In retrospect, I should have sent the game directly to the Microsoft Gaming Devices testing lab for their review.

The hardware problems with JSF prove that you can never be too complacent with your testing. Try to cover as many hardware permutations as possible. Test a featured peripheral with all if its drivers.

TOMB RAIDER II. This project was of paramount importance to the company. Therefore, we not only tested the game on our internal test bed system and our own developments systems, but we even enlisted the services of a local testing lab. In the end, we covered nearly one hundred different system configurations. We found several hardware incompatibilities that the developer (Core) promptly fixed. Mike Mchale and Mike Schmidt (the producer) worked tirelessly on a detailed readme to address some remaining technical issues and answer basic technical questions on the game.

The targeted user base for TOMB RAIDER II (TR2) was much broader than for JSF. Many first-time computer buyers would be buying TR2 for Christmas. These users often chose name-brand models that ran the gamut from sub-$1,000 bare-bones systems to Pentium II 300MHz screamers. This hardware disparity made it impossible for us to achieve our total compatibility goal. For us to come close to guaranteeing total compatibility, we would have needed to hire several testing labs working weeks on hundreds of configurations. This was a financial and time impossibility for us.

Therefore, we had to carefully choose our compatibility goals. We opted to run tests on middle-of-the-scale systems (P133-P233 MMX class systems), with an emphasis on consumer models. We also targeted the latest crop of 3D accelerator cards (nVidia, Voodoo Rush, ATI Rage Pro, Permedia, and others) and a selection of popular 2D OEM cards (Trident, S3 Virge, Matrox, and so on). Next, we ran sound card tests on products from Creative Labs, Ensoniq, Yamaha,

A Hypothetical Test Bed

This is a good template for building your test bed. As always, technology will render many systems obsolete in the coming months. System configurations should be bi-annually evaluated and updated. Employing a test technician, who could manage the test bed, burn CDs, and run automated testing would be ideal.

There area some areas that are often overlooked in testing. What happens if you code your game as a true 32-bit Windows 95 application that requires a typical desktop system configuration, and the user tries to run the game in Windows 98, Windows NT, or on a laptop? What if the system is below specification? What if the system is far above the recommend configuration? If you don't support or test these systems, *make sure your customers know*. Many of them will be using these configurations, and they'll expect your game to run flawlessly on them.

This is the configuration we used on DEATHTRAP DUNGEON.

CORE SYSTEMS

- Intel P166 \ 32MB EDO RAM
- Intel P200MMX \ 32MB EDO RAM
- Intel P233MMX \ 64MB RAM
- Intel PII 266 \ 64MB SDRAM
- Intel PII 300 \ 80MB SDRAM
- Intel Pentium Pro 200 \ 64MB EDO RAM
- AMD K5 166 \ 16MB RAM
- AMD K6 233 \ 48MB SDRAM
- Cyrix PR200MX \ 32MB EDO RAM

You might also consider testing on a Winchip CPU, a Cyrix Media GX CPU and RAM configurations from 16MB up to 80MB.

Motherboards:
Test on AT, ATX, Slot 1, and Slot 7 motherboards.

PERIPHERALS

Soundcards:
Test on ISA, PCI, wavetable, and motherboard-based soundcards.

2D Video Cards:
- S3 Virge DX/VX/GX
- Trident
- Tseng Labs
- ATI Rage II

3D Video Cards:
- 3Dfx Voodoo/Rush/Voodoo II
- PowerVR PCX/PCX2
- Rendition 1000/2100/2200
- Number Nine Revolution
- Matrox Mystique/Millennium II
- nVidia RIVA 128
- 3Dlabs Permedia 2
- Oak Warp 5

CD-ROM Drives:
Test on drives running on speeds from 4x up to 32x. Also try your game on IDE and SCSI drives, as well as CD changers.

Hard Drives:
Test on EIDE hard drives (including Ultra DMA) with motherboard and PCI controllers, as well as SCSI drives (SCSI2 and up) with motherboard and PCI controllers.

Controllers:
- Analog
- Analog (programmable)
- Digital
- Digital (programmable)
- USB

Speakers:
Test your game using low-end (those included in popular bundles) and high-end (preferably with Dolby Surround or AC-3) speakers.

Mwave, Videologic, and DiamondWare. Finally, we tested all the popular gaming control pads (from Microsoft's Sidewinder Pad to Thrustmaster). We decided upon this sampling through the invaluable research of technical support and marketing.

After the game was released, the waves of consumer calls that started to hit our overworked technical support staff were tremendous. The technical problems that we saw began with users trying to run the game on video cards that were incompatible with DirectX 5, causing the game to crash. Then we had users running the game at high resolution on low-end video cards with only 2MB of VRAM, which resulted in the game running slowly and showing severe texture loss. We had a problem with the game (which ran mainly from the CD-ROM) overaccessing the CD-ROM, which resulted in FMV problems. Finally, we even had some users who couldn't fully see the contents of the CD-ROM because they weren't running Windows 95 CD-ROM drivers. Thanks to our compatibility testing, we knew about the DirectX 5 and video card problems. We were able to document solutions in a detailed readme file. The CD-ROM issues required a patch to resolve the problems.

Although our attempts to prevent a flood of calls seemed to have failed, in retrospect, we actually did pretty well. We sold a substantial amount of product, but only two to four percent of the user base (about 4,000 out of 250,000) called with true technical problems.

By creating a profile of the typical user that is going to buy your game, you can tailor your compatibility testing to the type of system hardware that your game is most likely to encounter. In the end, your goal should be to limit calls to about two to seven percent of your user base. Generally, the broader the target market, the more calls you will have.

We all learned some lessons from this experience. First, beginning users often don't read or understand the technical documentation, so it's important to write clearly for easy comprehension. Second, a small percentage of users will have system configurations (such as three CD-ROM drives, SCSI and EIDE hard drives, Windows NT or Windows 98 beta, and so on) that are rarely used or recommended. These users are very hard to quantify, and their incompatibility issues are the hardest to resolve. Third, developers and publishers need to test more consumer systems. The vast majority of users are buying these systems because of their value for money. These systems are, to put it mildly, a compatibility challenge.

Test Bed Creation

As you develop your title, you'll no doubt be testing the game on systems with quality components, a fast processor, lots of RAM, and a superb 3D accelerator. Some of your audience will have systems of this quality and probably better. Nevertheless, most users will have systems that are not as powerful or designed for game use. Many users will have upgraded their systems with custom components. You'd better believe they're running every utility and tweak to gain performance on their system. The newbie game player will most likely be running a consumer system, designed to be easy to operate with everything a user would need. What the user doesn't realize is that most of the hardware components are either non-standard or outdated (read non-DirectX 5 compatible). With all these system types, what's a project leader or QA manager to do?

Windows Software

- Windows 95
- Windows 98
- Windows NT (up to 5.0)
- DirectX 5, DirectX 5a, and the forthcoming DirectX 6
- QEMM97
- Hard drive compression (Drivespace 3.0)

Also test your game while these common third-party programs are running in the background or are minimized:
ICQ, Virus scanner, Uninstall monitor, mouse software, e-mail software, joystick driver, video card configuration software, Powertoys, MS Findfast, MS Office Toolbar, MS System Agent

Multiplayer

At least eight systems should be connected via an IPX or NT network. Two systems should be connected via serial cable. Two systems should be connected via modem. One system should be designated as a primary dedicated server for TCP/IP games.

Miscellaneous

Invest in a quality test bed rack that can hold five to ten systems. You should also have several high-quality 17- or 20-inch monitor, six to ten systems switch box for monitor, keyboard, and mouse, and multiple joystick switch box

Testing Lab Checklist

Managing a testing lab and then coherently tracking incompatibilities can be a daunting task. However, here are some tactics to overcome potential problems:

- ❑ **Create a driver CD.** Burn a CD with all the major drivers for your peripherals. Include Window Cab files, virus updates, and third-party utilities. This CD will be invaluable if your need to reinstall drivers or system software.
- ❑ **Get a sturdy file cabinent.** Invest in a large, secure file cabinet that can store all your hardware, software, cables, and so on.
- ❑ **Create a system inventory.** Keep an inventory on all the systems in the office, including the test bed, that you could test on. In this way, you can find targeted systems without bothering MIS.
- ❑ **Invest in a video tracking system.** Whether it be a camcorder or an SVHS VCR connected to the monitor output, you sometimes need a video record to show to the programmer when describe a problem.
- ❑ **Create a multiplayer network.** Make sure the test bed systems are connected on their own hub for LAN and TCP/IP game testing. It is much easier to track problems when the testers are not scattered about the office.
- ❑ **Get testing tools into the game.** Tracking problems is much easier if you have testing tools (screen shots, performance tests/monitor, multiplayer log creation, and so on). These tools can help detemine whether that bug is hardware or software related.

JOINT STRIKE FIGHTER taught its testers to always be vigilant for strange hardware permutations.

The wide variety of consumer systems that TOMB RAIDER II's technical support staff encountered presented a compatibility challenge.

The answer is to create an internal test bed complied from your organization's market research. Market research entails retrieving data on what systems your customers use or plan to buy. There are two ways to get this information. First, you can have the QA manager review literature (online and magazine) on the hot new peripherals and system configurations. The second method is to review your publisher's registration card data. You should be addressing some specific questions:

- What is the typical system (40-60 percent of your target user base) that is used by customers? A "system" consists of CPU, RAM, CD-ROM drive, and video and sound cards.
- What vendor (for example, Compaq, Dell, or Micron) seems to be the leading consumer model? What's the leading model number? Any consumer model that has over five percent of the user base or consistently garners top marks needs to be in your test bed.
- What item is a "must have" for users or is something that they definitely plan to buy in the future? Try to get specific information. A "3D card" is not that relevant; an "nVidia RIVA 128 card" is relevant.

Plan to use components that account for a sixteen-month period from the ship date (eight months before and after). Complete the test bed with an updated compatibility checklist that is used to track the configurations and form your testing regime. In building your test bed, mirror the hardware used by both hardcore game players and the general public.

Gaming magazines and web sites are great resources for determining what hardware game players are buying now and what they will buy in the future. I've found that the CNET and ZDNET web sites are invaluable for this research.

Creating a test lab requires both space and money, but gaining a reputation for quality products is worth the expense and effort. If you are on a tight budget, you can get by with seven swap systems. I've listed a hypothetical test bed in the sidebar.

Allies in the Crusade

Use all of your available personnel resources to ensure that your game is compatible with your target hardware.

SALES AND MARKETING. Your company's sales and marketing departments can provide you with information on potential OEM bundle deals. If they'll be bundling your game or demo with an obscure video card, be sure to get the card and the SDK needed to complete the job. Sales and marketing also have a wealth of information on current and future demographics (including hardware specifications) of the gaming market.

THE HARDWARE VENDORS. Cultivating cooperative relationships with hardware vendors pays many dividends. First, you usually get free use of their current product lines and future use of any prototype units. Next, you get to send your beta to them for intensive tests on their product lines (free compatibility testing). If a problem should arise, they'll lend their technical expertise to solving it. Just make sure that you send them some free titles and give them credit when it's due.

TECHNICAL SUPPORT. These folks get the grief when products have bugs or hardware conflicts. From their first-hand experience, they are a repository of knowledge when it comes to potential hardware issues. They often have the answers to those obscure problems that can plague a game. Technical support has a vested interest in making sure your game works properly. Enlist them in the testing program. Technical support can casually test the game as a user would, spending perhaps 10-20 hours per person with the master candidate. Work with the technical support manager to assure that each system in the department has a unique system configuration. Use your technical support lab to emulate obscure video and sound card combinations.

QUALITY ASSURANCE. Basically, your QA personnel will be testing your game on its own uniquely configured systems. In the compatibility-testing phase, have QA operate its systems normally (that is, with typical background appli-

TABLE 1. *Saboteurs to your code.*

The leading culprits in hardware incompatibility are

1. **Third-Party Drivers:** Often you'll find users running alpha and beta versions of drivers or non-DirectX 5 certified drivers.
2. **Background Programs:** These include the mouse driver, ICQ, virus scanner, install monitor, joystick driver, and so on.
3. **Buggy Operating Systems:** Windows 95 buggy? Are you surprised?
4. **Video cards:** You're up against a market featuring many chips with myriad capabilities.
5. **Sound cards:** Compatibility issues involve legacy support, faulty game ports, slot specification, and so on.
6. **Joysticks:** You may have to consider sticks with 10 to 20 buttons, custom programmable software, and keyboard emulation.

These are your enemies. They are present in every game player's system. They wait to conflict with your game.

cations resident) as opposed to clean (with no programs other than Explorer in operation). With QA running your game on systems that are configured for day-to-day, practical use, you have the opportunity to catch many software incompatibilities.

As an additional benefit, your QA staff is savvier to the workings of the game and can more accurately pinpoint problem as coding errors or as hardware/software conflicts. The QA team also has the experience to duplicate hardware or software configurations that have created problems with past games.

Use them and trust them; they are there to protect your company from defective software. QA will be on point in the detection of conflicts. If you show them that you respect their time and effort, they'll give that extra effort needed to ensure a quality game.

THE USERS. Many solutions to difficult hardware conflicts come from industrious and clever users. In many cases, compatibility bugs in the release version were easily solved by users, who then posted the solution to a newsgroup or e-mailed it to technical support. This is often the case when the culprit is a missing driver, a corrupt .DLL file, or even a DMA conflict. If your users highlight a specific compatibility problem (such as ICQ messages crashing the game), do your best to address it. It must be important if they bothered to call.

Be open to their ideas and credit their assistance. If you have faith in your users, they'll spread the word that you listen and care about your customers. Such a reputation can only help you in the long term.

90 Percent of the Market

So, is global hardware compatibility an achievable ideal? Frankly, no. There are just too many hardware and software variables with which to contend. Nevertheless, we can limit the damage. Following a thought-out compatibility test plan (see "Maxims for Compatibility Success") can realistically verify that 80-90 percent of the systems on the market can run your software at the intended performance level. Ensuring stable, compatible games will earn you the respect and appreciation of game players. ■

Maxims for Compatibility Success

- Regardless of your talent and thoroughness, there will always be hardware conflicts with your game.
- Developing realistic hardware requirement specifications will alleviate the volume of conflicts. Do not give in to your marketing staff when they ask you to lower your specifications to gain customers. The customers you gain will be mighty angry when their systems run your beautiful game poorly.
- An excellent tenet is to start early. Many developers only request compatibility testing on the master candidate. This is dangerous. If you do discover a compatibility problem, finding a solution could jeopardize your launch date or necessitate a maintenance patch immediately after release (which equals very bad PR). I like to allot a two-week compatibility test run on the final beta version. The first week is to route out any problems. The second week is to verify any revisions.
- In the development schedule, add at least two weeks for compatibility testing and the resulting code fixes.
- Consult daily with the QA manager and the lead tester during this two-week compatibility test phase. Try to test the game in standard and non-standard environments.
- Include (with the assistance of QA and technical support) a detailed readme file that aims to circumvent any technical questions or concerns. Empower the user.
- Cultivate relationships with leading hardware vendors in every component field. They will gladly provide their hardware and knowledge in solving your problems. Send these contacts a near final beta of the game and ask them to test it on their systems. They are a great source of free compatibility testing.
- Contract with a local software-testing lab to assist in the testing of your title (for example, Veritest, ST Labs, Sys Labs). They are especially valuable because they have several swap systems designed to reflect user configurations. With their trained hardware technicians, they can pin down generic incompatibilities.
- Talk to your technical support manager about his or her experience with common user problems with this (your) type of game.
- Once the game is released, assign someone to monitor user feedback. If there are hardware problems, move quickly to solve them. Game players hate it when companies pass the buck.
- If a game is just not compatible with a hardware device, do the right thing and say so on the game box.
- The QA manager should have a testing checklist or documentation on how compatibility testing should proceed. If you have a long-term contract with a publisher with internal QA, press to have this document created.
- Functionality testing is the primary testing regime for compatibility testing. Functionality testing encompasses:
  - Game installation
  - Game play at options default for five minutes
  - Game play with option changes (especially video resolution) for five minutes
  - Game save
  - Game load
  - Game uninstallation

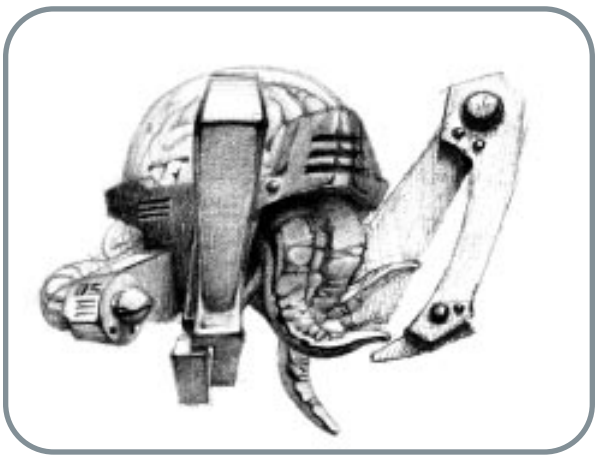
  Each tester should complete functionality testing within 25 minutes.
- Compile a compatibility chart that records the daily testing configurations and results and give this document to the lead tester at the end of the day.

The A of Lo -Pol go Modeli g

by Paul Steed

To say that creating low-count polygon meshes for computer games is an art form may be a little pretentious. But unless you've been given the task of creating a fully articulated humanoid character using only 100 triangles, you may not know how difficult this specialized form of model building really is. I've been modeling and animating for over six years and I'd like to share some of my techniques and poly-philosophy with you. Keep in mind that because I work at id Software making games such as QUAKE II, my discussions and examples will be culled mostly from that area of experience.

Today is the day of real-time polygonal games such as QUAKE II, WING COMMANDER: PROPHECY, TOMB RAIDER 2, and multitudes of other products touting 3D game-playing experiences. Whether on a PC, console, or arcade system, developers are using 3D models instead of prerendered sprites to represent the action; and those models need to be *lean*.



Pre-Modeling Phase

LIMITATIONS. The first key to creating low-polygon objects and characters successfully is to identify your limitations. This may seem a constricting and pessimistic approach, but it's crucial. Get a target face-count from your programmer for each character or object per situation. This information will save you time and frustration. For example, in QUAKE II we had around 600 faces per character — give or take a 100 — with which to work. However, characters that predictably appeared in relative droves (four to eight at a time) had to have lower than average face counts — about 450 faces. And a single boss character that had only one or two other character types had almost 2,000 faces.

Another face-limiting factor that you may have to consider as a low-polygon artist is the complexity of the character's environment or, in our case, the level. Get together with your designer and discuss how detailed the game level is.

After surviving adolescence and never quite growing out of it, the author likes to think of himself as an art Samurai. Primarily a modeler and animator serving time at Origin, Iguana Entertainment, and Virgin Interactive, Steed currently swings a sword for id Software, specializing in models, animations, and cinematics. Learning, growing, and teaching his craft is the cornerstone of his digital-bushido philosophy. Kicking ass in a good game of pool is pretty important, too.

More complex levels need lower face-count monsters and/or fewer characters. Be aware of all the variables that dictate your face count and use them to your advantage.

APPROACH. After you've determined how many faces you have to work with, you can start thinking about how to approach your impending modeling tasks. Let's go back to the 100-triangle humanoid figure. Does this figure need to uni-bodied (as in, all one unified piece), or can it be a group of objects intersecting and overlapping each other as in the VIRTUA FIGHTER games? In QUAKE II, I used a combination of both figure types with no problem.

Ask yourself some other questions as well. Which is more important, the total number of vertices or the total number of textured faces? For example, in QUAKE II I based my modeling tasks on the number of faces doled out to me by the programmers. However, I should have concerned myself with the total number of vertices, because our engine keeps track of vertex positioning as opposed to face positioning. I could have added a few more faces per model in QUAKE II had I thought to ask if hidden or submerged faces could be deleted. I just it took for granted that all the pieces of the models had to be "closed" like an airtight balloon. Now I know better; if you can't see a face on the model, it gets whacked.

TOOLS. I still use 3D Studio 4 (3DS4) because I literally can't make the time to learn something else. The other reason that I'm still using this dated tool is simple: it gets the job done and done quickly. I run 3DS4 on a Pentium 200MHz with 128MB RAM, a Monster 3D accelerator card, and a Matrox Millennium video card with 4MB RAM. You may use or advocate 3D Studio MAX, LightWave, Nichimen, or some other NT-based modeling package — what tool you use when you're doing low-polygon modeling really doesn't matter as long as you can get the job done. In the end, what's important is that you're able to understand and manipulate your mesh at the most basic level.

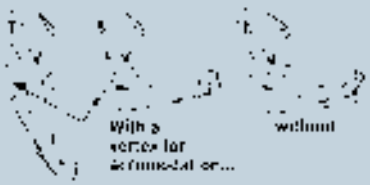
Any tool that you use will require you to "get your hands dirty" — you know, move those vertices and flip and turn those faces. So make sure that your modeling package of choice provides that type of functionality.

My first experience with low-polygon modeling was in 1992 when I did the objects for STRIKE COMMANDER at Origin. Using an in-house editor called EOR, I had no primitive-making command or lofting ability. I had to lay down vertex after vertex and play "connect the dots." As a modeler, this was the best thing that could have happened to me so early in my career. Creating models in this manner was tedious, but it gave me an intuitive knowledge of mesh building, and I highly recommend you try it sometime (schedule permitting, of course). Relying solely on an automated process for your meshes without hands-on vertex, edge, and face manipulation is like being a weapon-proficient soldier who has no martial arts or hand-to-hand combat training.

ACCURACY. I developed my low-polygon modeling skills making planes and tanks and buildings. Trying to make accurate-looking low-polygon objects is yet another part of the challenge. More often than not, you have to learn how to create an impression — as opposed to a literal representation — of a real-world object. For example, when I created some naval

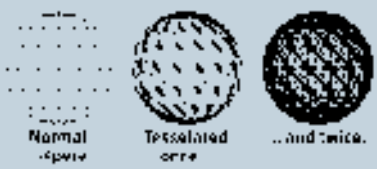
Glossary of Terms

Accommodation. What I call model "accommodation" is nothing more than making sure the model's geometry supports its animations. For example, even low-polygon limbs (arms, legs, tentacles, and so on) need to bend and flex while still holding their shape. When you make these types of appendages, create them in a bent state. I always see these stock models you can buy with their typical outstretched arms and legs frozen-halfway-through-a-jumping-jack looking pose. If the model has a million faces, then this type of pose is no problem. However, with low-polygon models, it's actually better to model limbs slightly or fully bent so the proper extra detail can be given to the elbow or knee. At the very least, manually bend these limbs at their intended joints to see if they hold their shapes and then straighten them back up if necessary.



Attitude. This is the life and identity shown in characters. Don't be afraid to give your creations personality. I exercise my imagination quite freely when imbuing my characters with virtual life. Giving them identity allows for better game play, in my humble opinion.

Tessellation. Tessellation is simply adding faces and vertices to an object for more detail. This is most useful with curved surfaces, but can be used for any shape. Of course, with low-polygon objects, the last thing you're likely to do is tessellate an object because it adds even more faces, which you're trying to find a way to get rid of.



Boolean. You'd think that Boolean would be the best of tools for creating models because it alters the shape of an object by using the intersection, subtraction, or union of another overlapping shape to subtract or add to the original object's geometry. In reality, this tool can be problematic because it creates extraneous faces that are sometimes hard to find (at least in 3DS4), thus adding to your face count unnecessarily. Still, Boolean is very useful when it works the way you want it to work.

Curve angle. Curve angle is what I call the angle made by meeting segments in the edge of a low-polygon curve. If the curve angle is

Continued on p. 64

warships in STRIKE COMMANDER, I had to suggest the complexity of the masts, wires, and loads of antennae that are found on these vessels. Using opacity, I textured the geometry onto two quads and then crisscrossed them so that it looked like there was some busy conning tower present on the ship. There was no way I could have built all those structures with the limited number of polygons available.

When creating characters, the challenge of accuracy is even greater due to the complexities of simulating organic objects and allowing for realistic movements. Muscles are smooth, not blocky. Faces are usually somewhat round, not square. And then there's hair — don't even get me started on that subject. Even when you have many polygons to work with, creating hair accurately is a substantial feat. It may be best left to some kind of procedural or particle system. The bottom line is that anything round, rounded, or curved will eat up faces quickly.

RESOLUTION. Another factor to consider is the required level of resolution of your model. With regard to the ship that I described earlier, I knew the model would be seen while flying quickly past it, perhaps as the player tried to sink the vessel. Given that known low-resolution requirement, I concentrated on elements (such as the mast area) of the object that made it easily recognizable when displayed in only a few pixels. After technological constraints, resolution should dictate how much detail you need to put into your model. Obviously, close-up character models will need more faces than angular, distant battleships.

Before I move on to the following tutorial, read over the "Glossary of Terms" sidebar. I'm assuming that you're knowledgeable in basic modeling, animation, and art terms.

Modeling a Quake III Monster

Let's create a low-polygon monster that will be used in the upcoming QUAKE III. As in most artistic endeavors, we start with a sketch. Nothing too fancy is required — usually just an action pose and some close-ups of the trickier areas. Sometimes, a full-blown orthographic breakdown is necessary,

Glossary of Terms (cont.)

large enough (as in, closer to 180 degrees or flat), I'll reduce the segments or facets that make up the curve of the shape by merging or welding vertices. Therefore, the higher the curve angle, the more likely the chance a shape can be optimized.

Diamonds over squares. When you need to make a tube-like shape such as an engine nozzle or cable, a pentagon is the optimal shape for circular low-polygon objects. But sometimes we need to use less than a five-sided segment, and a square or triangular cross-section is all we can afford. If the shape is supposed to be round, however, make it a diamond instead of a square. The reasoning for this may be debatable, but it's been my experience that the object will appear more rounded with an edge along the top instead of a flat end.

Edge Division. This command is the opposite of the Vertex Merge. It is the process of dividing an edge exactly in the middle, inserting a vertex, and creating the appropriate faces resulting from the new vertex's addition.

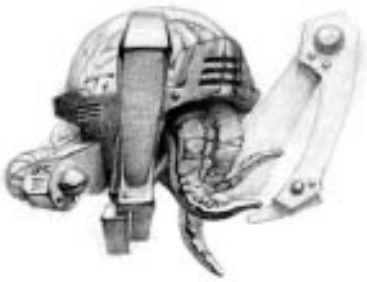
Edge Turn. In PowerAnimator, it's called the Quad Split or something, but basically this technique takes the bisecting line of a quad made up by four vertices and turns that edge so that it goes to the other two vertices. It's a very useful tool, and I wouldn't touch a modeler that doesn't have it.

High to low vs. low to lower. These are two methods by which I model based on how I'll be using the mesh. The former method is usually reserved for meshes that will be in high-resolution cinematics as well as in the game. Initially, an unlimited face-count approach is taken, and the model will end up being very detailed. This high-resolution version can be used for advertisements, cinematics, or whatever. It also serves as the template to model a low-resolution version of the same model to go into the game.

The latter technique involves creating a mesh without considering a high-resolution version and keeping the face-count relatively low from the beginning and optimizing on the fly. Sometimes though, I'll make a high-resolution version of a character or object just to have a starting point to texture. Usually, Adrian is our texture guy, so I don't worry too much about the texturing process.

Level of detail (LOD). This is when an object is represented by varying numbers of triangles at varying viewing distances. For instance, at long range an object may be no more than a few pixels on the screen, and the closer you get to it, the more detail pops into view. This technique can be done manually, but the more levels of detail you create, the bigger the memory hit. Each LOD is a separate object that has to be stored and tracked. A technique called "displacement mapping" or "real-time deformation and tessellation" based on displacement map information can overcome these LOD limitations, but has yet to be implemented on a large scale.

though. In this case, we have a monster design from fellow artist and id founder and co-owner, Adrian Carmack.



Checking with Kevin Cloud, fellow artist, project director, and co-owner of the company, we determine that this particular guy (called an Exterminator) is halfway between a boss and a basic monster. So we set the face limit for this guy at 800 (not a small number for a monster) because he'll be seen rather infrequently. Of course, anything less than that is better, but I know we have at least 800 faces with which to work (and generally, if I'm given 800 faces, I'll *use* 800 faces).

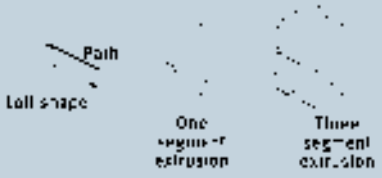
We're taking a "low to lower" approach to building this guy, but we don't have to think too frugally at first. Generally, it's best to use whatever's necessary to get the shape going (within reason) and optimize as you go.

PART ONE: THE ARMS. Having all the premodeling information we need, let's start with the arms. The reason we start there is that it's easier to begin a model with its most distinguishing feature. Thus, we pick the arms. Because the shape is rather unique, primitives won't work very well. So let's create an outline of an arm and loft it along a path.

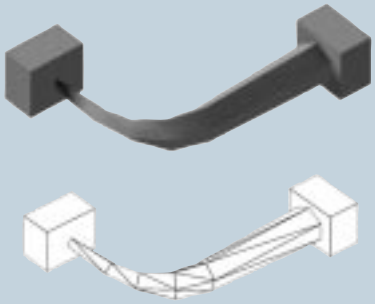
Shapes

Since it's bulky and organic and needs to appear somewhat beefy, the loft of the first arm needs two segments

Lofting/Extrusion. Taking the outline of a 2D shape and lifting it into the third dimension along a path is called lofting or extruding that shape. This technique is very useful when a primitive won't fit the bill. Lofts can have as many segments or layers as you require.

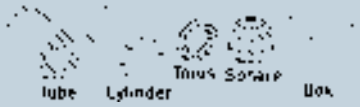


Mixed cross-sections. This is a technique whereby a shape is not constrained by the same cross-section or lofting shape in its length. This works especially well with darker or smaller objects, or when a cross-section is very noticeable in the design and needs to look rounded.



Optimization. Optimizing your model is the act of reducing the number of faces making up that model. Almost every modeling tool has some sort of optimization feature. Some are good, some aren't so good. I'm a neat freak with my models, so I try to keep some sort of order or symmetry to the design of the mesh whenever I can. A lot of optimization programs aren't so aesthetically-inclined (to say the least). Thus, I always prefer to optimize by hand when I do low-polygon meshes (unless the mesh needs to go from 10,000 faces to 500), because I have more control. When I do resort to automated optimization, I'll usually do it by splitting the model in half (provided it's supposed to be symmetrical), optimizing one half, and then welding the two pieces together.

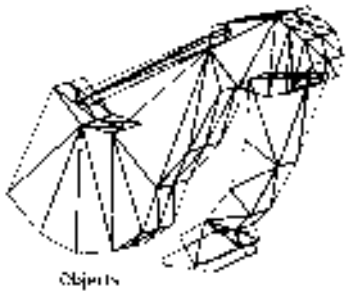
Primitives. Most modeling packages include a group of polygons that can be created quickly via a single command such as Create⇢Box or Create⇢Sphere. Primitives are excellent building blocks for modeling. If primitives can't fit your shape needs, then of course a loft or Boolean would be the next approach.



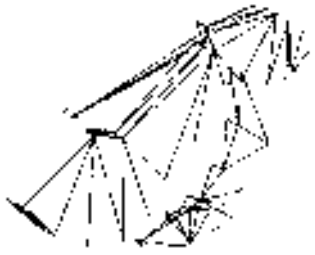
Splitting the difference. Sometimes, you'll get a shape or a section of a shape that needs to be represented by a more triangular shape than a rectangular shape. A quick and easy way to accomplish this with accuracy is to split the difference. Basically, you take an edge, divide it, and merge the end vertices to that point. This technique merges two end vertices precisely in the middle of an edge. You could just merge the vertices and move them, but then that'd be more work now, wouldn't it?

Vertex Merge. To me, vertex merging is the best friend I have when it comes to creating low-polygon models. It is key to ridding yourself of unwanted faces and vertices, allowing you to hit that magic number given to you by your programming staff or art director. Merging a vertex simply takes one vertex and merges it with another vertex, effectively reducing the number of vertices by one and the number of faces by two in the immediate area of the merge. This technique or command, combined with Edge Division and Edge Turn, form what I call the "Trinity" of most useful low-polygon modeling techniques.

(normally, one is best with a low-polygon object). Lofting the shapes results in an initial 200-face arm assembly, which we need to scale and position to match the sketch.



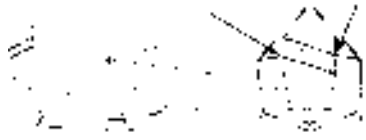
Doing some quick math, a 200-face arm isn't good because there's two of them, and we still have to create some articulated tentacles, a rounded head, and a body structure. Budgeting 400 faces for just the arms simply won't work. So first let's chop off any faces that won't be seen. Remove the ends of the tendon at the top, the top faces of the lower arm, and the base of the arm where it will connect to the body. We also need to rotate the tendon so the edge is up and exercising the "diamonds over squares" rule.



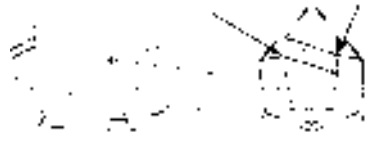
Now things start to move pretty fast. We split the difference on the bottom edge of the claw and merge the vertex at the upper end of the tendon (mixed cross-sections), making sure that we turn the bottom edge of the tendon for clarity and neatness.

Then we move the outer-edge vertices of the main arm object to give the top and bottom an edge suggesting roundness (diamonds are better…). Merge some vertices at the top of the main arm and go ahead and insert a vertex by dividing an edge at either side of the base of the main arm. Now pull them out to give the arm a "turkey leg" look. Given the current curve angle of the bottom of the main arm, we need to merge some vertices by splitting the difference in two places. Then merge the vertex in the middle, underneath the "neck" of the arm. Grab two vertices at the top of the end of the arm and pull them down and scale them out. Then go nuts at the end of the arm where the lower arm attaches. Merge and turn and move and scale the hell out of that area. Here's a comparison of where we started and where we are at this point:

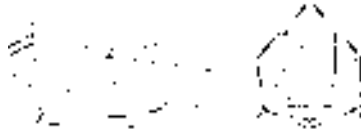
Now look closely at this blow-up of the end of the main arm object. It should be obvious how we can knock nine faces off of this area. Which three techniques do we use?



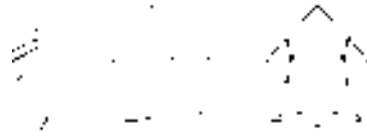
If you answered, "Why Paul, the curve angle underneath and to the right of this region easily dictates optimization by nine faces by taking these two edges…



…splitting their difference…

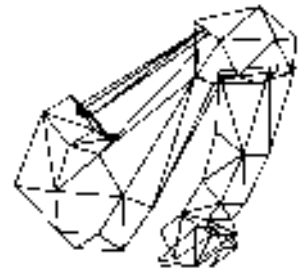


…and merging the vertices,"



then, voila. You are absolutely correct.

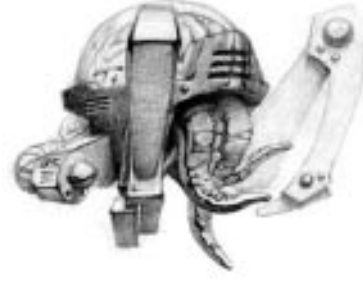
By now we're feeling pretty good about the arm, so we scan it for more tweakage. (We're down to 131 faces in case, you haven't been keeping track.) Glancing at the lower arm, we see via our hyper-keen curve-angle sense that a couple of faces can be shaved near where the claw is joined.



Why not get rid of the vertices bisecting the leading edge of the lower arm, too? We can still imply a slight curve to the arm with the inner vertices. Doing this, we also realize that the edges look better turned because, well, it just looks better.

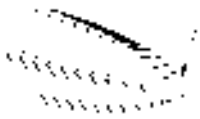
Now we're down to 123 faces; good enough to move on. All in all, that took maybe 25 or 30 minutes to complete.

PART TWO: THE REST. The next most easily do-able part of this monster is the tentacles. Let's look at the sketch again:

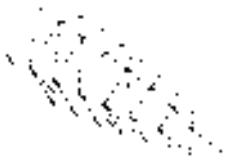


There are three of them, and they look as though they wave around like an elephant's trunk, so they need to be articulate. Of course, they need enough segments to accommodate their animations, so we opt for lofting a pentagon with, say, 11 segments.

That looks good, but a single tentacle weighs in at 116 faces (which, after adding the other two, is *too* many faces for comfort). So after deleting the faces at the base, merging the five vertices at the tip (a point is fine here), and making the last segment into a square cross-section, the tentacle is starting to shape up.



Still, 103 faces for one tentacle puts us at a current total of 555 faces without the guns, the hose connecting the guns, the head/body, or the brain case. Uh, not good. Let's get rid of the top three segments of this rascally appendage and see where we are.



Hmm… 73 faces. Sold! We'll sit with that for the time being and move on. But first, lets adjust the tentacle mesh to accommodate the animations better. From experience, I know that the majority of the animations for this type of appendage take place in the bottom two-thirds of the object, so I'll adjust the segments incrementally to support this. Let's also make it less smooth and slightly more wavy.

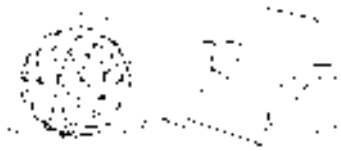
Notice that we've built the tentacle to hang straight down. Rather than bending it to ensure accommodation, we're going to rely on bending and undulating the thing in the animation tool with a skeleton inside the mesh. Now to the body.

In 3DS4, we can start the body with a primitive called a g-sphere, or geodesic sphere. An l-sphere, or latitudinal/longitudinal sphere, would be cleaner than a g-sphere, but this is one of the few time a less symmetrical approach is better. Actually, g-spheres are better for round objects when doing low-polygon modeling because we get more bang for our buck. Also, a g-sphere can be specified by total number of faces, as opposed to segments. Here's the difference between a 144-face g-sphere and a 14-segment (diameter) l-sphere:

They look about the same in resolution, but if you look at them top down (with the g-sphere's top edges turned as a modest concession to symmetry) you'll see that the g-sphere actually has 15 segments in diameter while the l-sphere only has 14 segments. This is even more significant when you consider that the l-sphere has 168 faces vs. the g-sphere's 144 — more rounded diameter with fewer faces.

So let's start with the brain case. It can be a simple dome because the body comes up over it around the edges. Delete the vertices beneath the equator of the sphere because the deleted faces underneath won't be seen. Oops — first we need to make a copy of the sphere so we can use it for the body as well.

Now let's do a Boolean operation and make the shape that we want for the body. First, we have to create the Boolean or cutting shape. We do this is by drawing the outline of the shapes around our sphere and then lofting them into objects with which to cut.

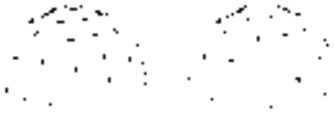


Make sure that the Boolean shape intersects the sphere completely and then use it to "Boolean," or cut away, the parts of the sphere that it touches.

Keep in mind that doing a Boolean operation means that the object used to cut will be deleted. Make a copy of it if you're unsure whether or not you'll need it again.

It's time to start optimizing again. Let's start with the dome on top the body. We've already deleted the faces underneath, so we don't have to worry about that. Since the character won't be seen up close in a nonattacking manner, we can shave the top down a little and merge the very highest point. We can also use the high curve angle in the front to merge those faces (for the most part, the monster will attack head on, so a profile can be less detailed

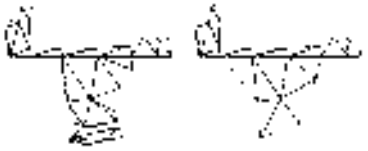
than a frontal silhouette). Merging these vertices results in



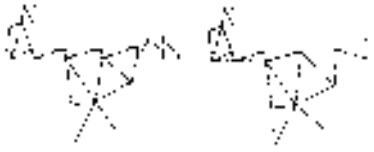
Now let's look at it from the top and pretty it up a bit. Turn the top edge in the middle. The curve angle in the back is pretty high, so we'll split the difference and merge those vertices. Let's tweak the shape a little by scaling the outlining vertices. I see some vertices here that can be merged. What do you think?

Excellent. We went from 75 to 62 faces.

On to the main body (which has 126 faces). Let's begin by simplifying the bottom area and front and back mouth area. The bottom will be obscured quite a bit by the tentacles, and the front and back areas will have objects coming out of them. So let's start merging vertices at the edges of these areas.



We just knocked off 34 faces. We're cruising right along, but that narrow segment near the top of the body bugs me. Lets simplify it with some more vertex merging. Also, the back looks overly complex. Because it won't be seen too much by the player in the game, chop it up.



The body's front area is supposed to look like some sort of hard metal holding edge, but I bet we can merge those side vertices, leaving the top alone, and still get the impression of hard metal.

So, weighing-in at this corner, we have a newly optimized body consisting of 68 faces. Overall, if we have three tentacles, two arms, a dome, and a body, we've used 595 faces. This leaves us plenty with which to do the guns and connecting hoses. Speaking of which…

Wait a second. There are still two faces that can be deleted.

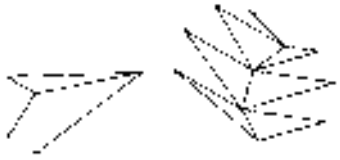
They're hidden by the front plate of the body, so they can go.

All right — we're nearly finished. Before we build the guns and tubes in the back, let's position the pieces to approximate the sketch better. I need to scale the arms up a little and make the ends of the arms look knobbier.

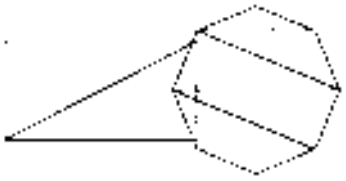
Close, but the lower arms need to be heavier-looking. And a little bigger…

Good enough. Let's build the guns.

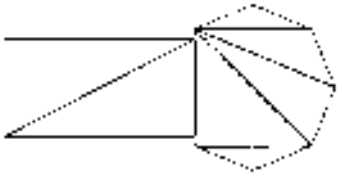
Let's use a cylinder primitive to make the barrel of the gun and a box to make the connecting arm of the barrel. We'll use a torus to make the cable in a minute. We'll begin with an octagonal cylinder and then make the box.



Rotate the cylinder and move the box over.



Delete the end faces of the box since they'll be hidden. Merge the two vertices closest to the box on the cylinder to save a couple faces and viola. We have a gun.

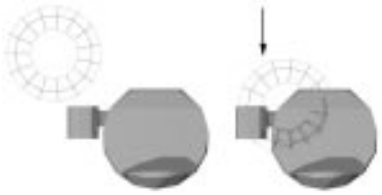


Let's move it into position on the monster…

…and make sure that it will accommodate its animation (swinging up and swinging down to fire).

Looks good. If we double the gun's face count to account for the gun on the opposite side, the total face count is now up to 657. We can make the connecting hose now.

Let's hide everything but the body and the gun. Create a torus primitive sized to matched the body and back of the gun barrel. We begin with a five-sided, 15-segment torus because we'll only be using about half of it. Then move it into position to the rear of the gun.



Because we only need the part of the torus that goes from the back of the gun to the rear of the body, delete the appropriate vertices.

We only have 70 faces to work with — to stay under 800 faces, we can only use another 143 faces (70×2 = 140). Checking the count on the hose, it has 80 faces. So…

…we delete the end vertices going into the back. The count is now 70 faces. Did I mention how good we are?

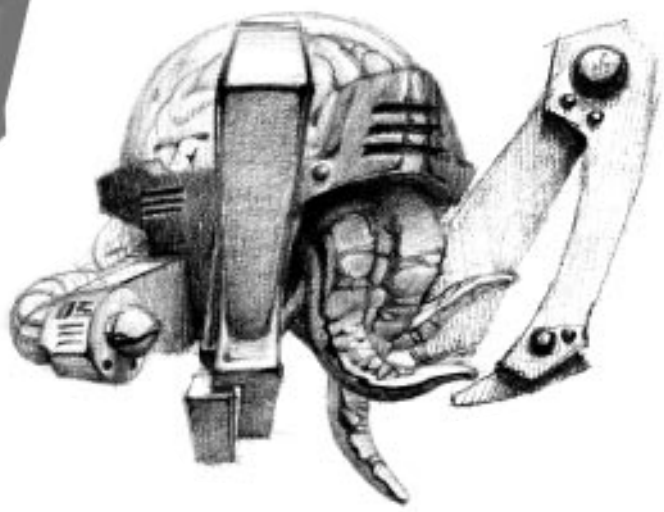
Our model looks good in top view, but let's make sure it's matched up in all three views. From the back it looks like we have to rotate it down a bit…

…and bend and move the vertices at the end of the tube so it goes into the body.

Works. Now let's copy the gun and hose and flip it to the other side and guess what?

We're done.

So we have a finished monster that's 797 out of 800 available faces, which in real-world time would take about two hours to create. ■



DeBabelizer Pro 4.5

by Greg Hammond

The phrase "Just DeBabelize It!" is not just a marketing slogan for Equilibrium. It has become part of the common language spoken by game developers everywhere. Need a bunch of graphics converted from 24-bit to 8-bit color using one palette that best represents all of the images? No problem. Need to stamp a copyright notice on each frame of an animation so you can publish it on your web site? No problem. Need to clean up bluescreen video and automatically composite actors and backgrounds? No problem. Those are the kinds of tasks DeBabelizer Pro eats for breakfast.

How many times have you converted graphics manually by opening each image file, clicking Save As, and then selecting the appropriate file format, all after mapping the colors down to 8-bit or some other equally pedestrian machination? Long ago, in the dark ages before I started using DeBabelizer, I can remember having to go through all sorts of gyrations to get palettes to behave and files to convert (not to mention trying to do anything "unusual" to the files, such as applying filters). Granted, at this point in time there are many programs, both commercial and shareware, that can run batch processes on groups of images, converting them to various file formats. However, most of these can't handle video or animation files, and none of them even come close to covering the scope of features found in DeBabelizer Pro.

During the course of preparing this review, I used DeBabelizer Pro 4.5 on a number of projects at Simutronics. I used the program's SuperPalette functions to map all of the front-end images for one of our adventure games down to a single 8-bit palette. When we needed to prepare animations for use on our web sites, I was able to process them down to the Netscape palette and add copyright information to each frame. I've performed countless other tasks with DeBabelizer Pro that would have been impossible (or at least, extremely time consuming) to accomplish with any other tool.

Greg Hammond is the art director at Simutronics Corp. He has been working in the entertainment software industry since 1981, when there were six colors and no one complained. You can reach him via e-mail at gregh@simutronics.com.

How Does It Work?

The core functionality of DeBabelizer Pro is the BatchList — quite simply a list of image files that can be sorted, saved, and processed as a group. Multiple groups can be organized within one BatchList, and the source files themselves can be located anywhere on the user's computer (or network, for that matter). BatchList creation is a snap: individual files, selection sets of files, or entire directories can be dragged from an open Windows Explorer window to an open BatchList document.

Scripts are a powerful means of automatically applying a number of actions and effects to an image or group of images (such as a BatchList). Scripts are great for controlling bulk image processing jobs, and the user can run multiple BatchList operations in succession without any intervention. Scripts can be nested within other scripts and stacked for running multiple scripts from one master script. This system allows for

tremendous flexibility in how the user chooses to process graphics.

There are two easy ways to generate scripts in DeBabelizer Pro. The first is to experiment with applying effects and operations to an image and then copy the desired steps from the image's log into a script. Each time an image is altered in any way within DeBabelizer Pro, that image's log is updated. Image logs allow the user to experiment by applying various effects to an image, and then copying the log of those effects to a script. This method is much more interactive than building a script by hand. Another quick method of script creation is to turn on the WatchMe feature, perform the required steps to process a sample image, and then save the resulting script.

Palette operations represent the essence of DeBabelizer Pro. Since most of us still have to deal with paletted images, it's good to have a tool like this. Palettes can be edited in many ways, and DeBabelizer Pro allows the user to perform palette manipulations that other programs can only do with unpaletted (16- or 24-bit) images. One of DeBabelizer's best features is the ability to mark certain colors in a palette, such as the default Windows colors, as off-limits — more on this later.

DeBabelizer is perhaps best known for its SuperPalette technology. SuperPalettes are used to create a custom palette for a series of image files. By right-clicking on an open BatchList and selecting Create SuperPalette, the user can automatically build a palette that represents all of the images in the BatchList. If the user needs a palette that is stronger in some colors than others, this is easily accomplished by weighting the BatchList with multiple instances of a source image with lots of the desired color.

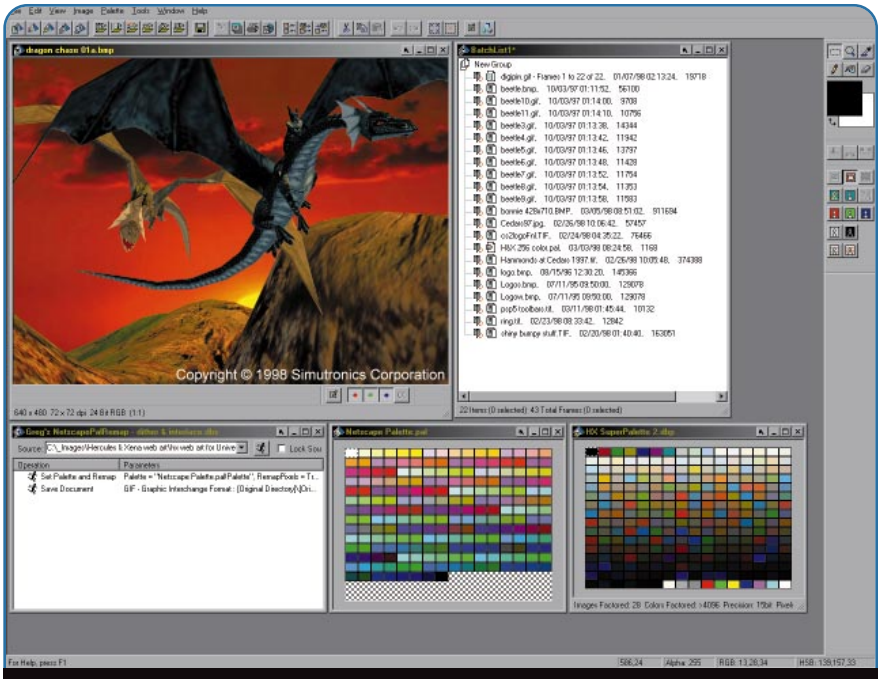


FIGURE 1. *The DeBabelizer Pro 4.5 workspace with Image, BatchList, Script, Palette, and SuperPalette documents open.*

What Makes DeBabelizer Pro So Good?

DeBabelizer Pro provides a comprehensive approach to processing graphics. The program reads and writes over 90 image and animation file formats. The support for each of these formats is really complete. I get tired of graphics utilities that can't read certain versions of .TIF files, for example.

Although it's definitely not a paint program, DeBabelizer Pro features an impressive array of image processing tools. DeBabelizer can scale graphics with a variety of algorithmic resampling techniques. It can swap, rotate, and convert channels to grayscale. And users can apply Photoshop-compatible filters to images.

One of my favorite tools in DeBabelizer Pro is Filter Interpolation, a feature that lets the user change the numerical values in a plug-in filter's settings over time. This is most useful for producing effects for animation and video. Unfortunately, only plug-ins that have an FltD resource will work with this tool (so Kai's Power Tools and other powerful plug-in systems are not supported — bummer). Still, game developers can do a lot of cool stuff with this feature alone.

Hey — I Already Own DeBabelizer 4.0! Why Would I Want to Upgrade?

Some of the new features in version 4.5 are well worth having, even at the $100 upgrade price. Enhancements to the BatchList give it more sorting options. Separate folders can now be maintained for better organization of large groups of files within the same BatchList. In addition, multiple batch processes can now be run in succession without any user intervention — very handy for those of us processing video clips.

DeBabelizer Pro 4.5 includes a number of enhancements to its multiframed image handling and processing capabilities. Users can now process video and multiframed files identically to single image files — the program treats both types of multiframed files as "Animations." Version 4.5 now saves an .AVI file to any file format supporting multiple frames, including .GIF animations and vice versa. It also generates multiframe image files from BatchLists, which users can now reorder in any sequence. This lets users create animations from any sequence of image, animation, or video files and save them to any one of the supported animation and video formats.

Several other enhancements in version 4.5 are also a big help for video and animation. The Blue Screen Removal tool now has a complementary pair of tools: Shave and Outline. Shave is used to automatically remove antialiased "halo" areas around objects and actors filmed in front of a blue screen. Outline does just the opposite, and can add a stylized visual emphasis to an object or actor. The Composite

FIGURE 2. *A photo of a model against a blue screen – the model's wispy blonde hair is a classic problem for blue screen photography.*

FIGURE 3. *The same photo after being processed by DeBabelizer's Blue Screen Removal tool. A background pattern and drop shadow were added.*

tool lets users easily create layered animations by placing any image file on the active image. This process can be automated by using the new Batch Composite command, allowing many frames of animation to be processed together. Batch Composite can also generate a script that users can reuse and modify.

Another animation tool added to version 4.5 is Pixel Shift. This feature shifts the contents of an image in any direction within the image's boundaries. This is a particularly useful feature for some game developers, as well as animators and digital video producers, because it allows the user to create a scrolling background for an animated object automatically. The entire image can be shifted and wrapped around, or the unused area can be filled with the background color. The Hanna Barbara animators would have killed for this tool years ago.

Documentation

The documentation, especially the User Guide, is very well done. The User Guide includes introductory sections about working with digital color, screen resolution vs. print resolution, and other helpful topics. This book is chock full of screen captures illustrating the program's many functions. Also included is a Quick Reference Card showing all of the toolbars and their associated icons.

A small Getting Started book provides an introduction to DeBabelizer Pro's features, as well as three very helpful tutorials. The first lesson covers the fundamentals, the second is about palettes and color reduction, while the third lesson is essential to understanding how the product works with animation and digital video. Equilibrium provides the supporting image files for each of these tutorials, and after completing all three, the user can confidently use most of the program's features.

User Interface

Equilibrium has refined DeBabelizer's user interface considerably over the last few years. The best evidence of this is their original interface device, the ActionArrow. These arrow icons appear at the top of each open document window. By dragging an ActionArrow from one document, say a palette, to another document, such as a BatchList, the properties of the source document are automatically applied to the target document. ActionArrows can save a tremendous amount of time over the course of one project, as compared to the typical dialog boxes that would be required to accomplish the same tasks.

The toolbars are fully customizable — in fact, there's an icon available for virtually every function in the program. Most users will probably find it more efficient and intuitive to access the context-sensitive menu systems via a right-click over an active document window. Also, some of the icons are so abstractly designed that, unless the user really wants to spend a lot of time becoming familiar with their meaning, he or she would be much better off using the menu system anyway.

One of my complaints with this product is Equilibrium's decision to prominently feature their pyramid logo behind each of the New command icons — New Image, Palette, SuperPalette, BatchList, and Script. DeBabelizer's icons are big enough at 23×22 pixels, but these primary icons are hard to read because of their cutesy design. DeBabelizer is somewhat different than other programs because it deals with six unique types of document (Image, Animation/Video, Palette, SuperPalette, BatchList, and Script). However, the company could have made the icons for each of these larger and then just added a sparkle or some other simple graphic effect representing the concept of "create a new file of this type." These icons are really problematic at the higher screen resolutions that today's game artists are likely to use.

Annoying Little Bugs

DeBabelizer Pro's basic paint functions are represented by the standard radio–button-style interface. However, during certain operations (for example, when using the Blue Screen Removal tool), when clicking between the selection tool, magnifying glass, and eye-dropper buttons, the program gets into a state wherein the user has to manually deselect these buttons to use one of the other paint tools. These are such basic functions for this program that it's quite annoying for them not to work properly — it points at what must be a fairly limited testing program at Equilibrium.

Remember that great palette editing feature I mentioned earlier? The one that lets you mark colors as off-limits (such as the default Windows colors)? Well, it's a terrific feature, but it currently has one nasty little bug. The only way to maintain the position of off-limits colors when a palette is sorted is to use the Specify option in the palette sorting submenu. But that's not the bug (even though it would be nice if DeBabelizer Pro kept track of the off-limits colors when using any of the other 12 sorting options). The bug happens when the user clicks the check box marked "Do NOT touch off-limit colors," and then proceeds to sort the palette. Sure enough, the program keeps the off-limits colors in their

proper place, but the flags that indicate their off-limits status are turned off. So in order to save the palette properly or to do multiple sorting operations on a palette with off-limits colors, the user is forced to go back and turn on the off-limits flags on those colors. If the user isn't paying close attention while sorting the palette, this subtle problem can be missed, leading to palettes that have the off-limits colors sorted into the mix — a Bozo no-no when developing for Windows.

On a positive note, the customer service and technical support people at Equilibrium were very helpful. I threw a few questions at them to test their knowledge of the product, and they passed with flying colors. Equilibrium seems to be quite responsive to their customers' needs and requests, and I was encouraged to send e-mail regarding two of the problems I mentioned to them.

The bottom line is this: I cannot live without DeBabelizer Pro. It simply does things with images and animations and palettes that I cannot do with any other program. The automation features (BatchLists and scripts) are true time and sanity savers. The addition of more video and animation tools in version 4.5 is enough to make the upgrade worth the cost for owners of version 4.0. The few bugs I've encountered are indeed annoying, but they are certainly not serious enough to keep me from using the program every day. I highly recommend DeBabelizer Pro 4.5 to anyone working with lots of images, animation, and/or video. ■

DeBabelizer Pro 4.5

Rating (out of five stars): ✪✪✪✪☾

Equilibrium
Sausalito, Calif.
(800) 524-8651 / (415) 332-4343
www.equilibrium.com

Price: $599.95 (Retail); $399.95 (Direct); $375 (Street); $99.95 (Upgrade from version 4.0)

Software Requirements: Windows 95 /Windows NT 4.0

Hardware Requirements: Minimum 486, 16MB RAM, 20MB hard-disk space

Technical Support: 90-day telephone support

Money-Back Policy: 30-days money back

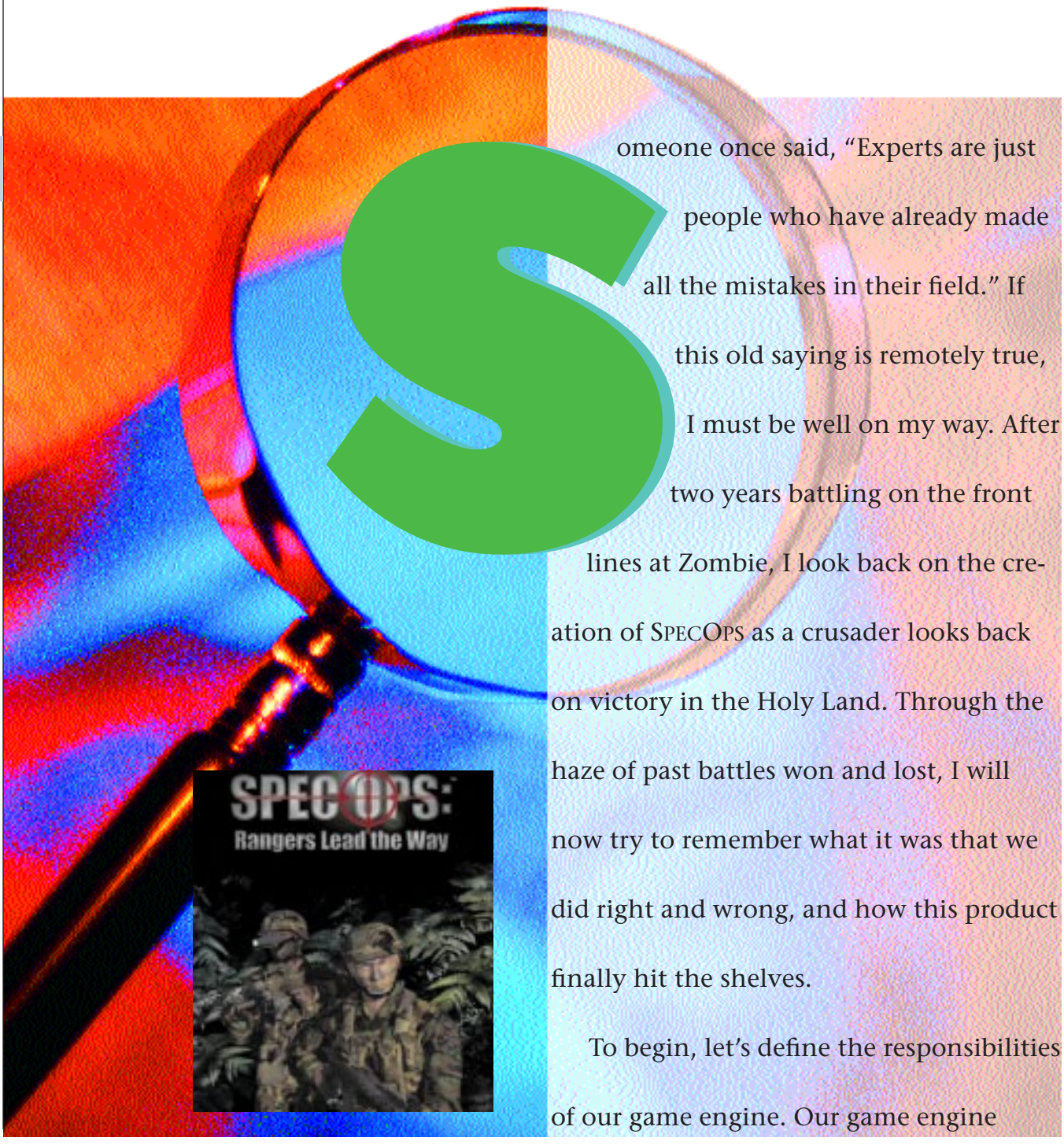
Pros:
1. The industry standard by which all other graphics conversion products are measured.
2. Extensive support for many graphics file formats.
3. Intuitive user interface facilitates rapid setup and execution of complex tasks.

Cons:
1. Designs of primary icons are hard to understand due to Equilibrium pyramid behind each one — especially difficult at higher screen resolutions.
2. Palette sorting functions don't track colors marked "off-limits."
3. Support for .MOV files still pending Apple's release of QuickTime 3.0.

Competitors: JASC Image Robot, JASC Paint Shop Pro, Crayon Software Magic Viewer, Adobe PhotoShop 4.0

Zombie's SpecOps: Rangers Lead the Way

by Wyeth Ridgway



Someone once said, "Experts are just people who have already made all the mistakes in their field." If this old saying is remotely true, I must be well on my way. After two years battling on the front lines at Zombie, I look back on the creation of SpecOps as a crusader looks back on victory in the Holy Land. Through the haze of past battles won and lost, I will now try to remember what it was that we did right and wrong, and how this product finally hit the shelves.

To begin, let's define the responsibilities of our game engine. Our game engine

manages many aspects of the game: sound, graphics, physics, game core, and others. Taken alone, these components don't do anything — they're just tools. The game-specific AI uses these tools to perform logical game actions. In this manner, a complete game is composed of the game engine plus the game-specific AI.

As such, the SPECOPS game is composed of the Viper game engine and the SPECOPS AI and resources. In this article, I will primarily focus on the design aspects of the Viper engine, which was the first and most important step in creating SPECOPS. In Table 1, you'll find a description of the project and its development environment. With these design requirements in mind, let's take a look at how each of the major components of the game engine was implemented.

Game Core

The game core provides an interface for all of the game engine components. Among other things, it defines and manages "objects" in the engine and allows them to be passed between components. For example, the game core might first send an object to the physics engine to be moved and then to the graphics engine to be drawn.

In the Viper engine, we chose to implement the objects hierarchically. Figure 1 shows a small portion of this hierarchy. The hierarchy allows memory optimizations because objects only require allocation of the structures that they use. It also provides some object-oriented structure to the program. In C, there was one .C file for each object type, with the same .H relationship as in Figure 1.

The only objects that the game core referenced and modified were those at the top of the hierarchy: **`Generic Object`**, **`Static Object`**, and **`Dynamic Object`**. The rest of the hierarchy is composed of game-specific data structures (such as the **`Character`** class). The support modules for these were separated from the rest of the code, so that it would be easy to remove them from the game engine code. This helped ensure the separation of game-specific logic from the game engine.

Zombie's SPECOPS development team.

This design was expandable and easy to work with. While building the engine, we could separate out the SPECOPS-specific code to create demos and even other games. Doing so at several points during the development cycle forced us to continually clean up any breaches of the design and maintain the reusability of the engine.

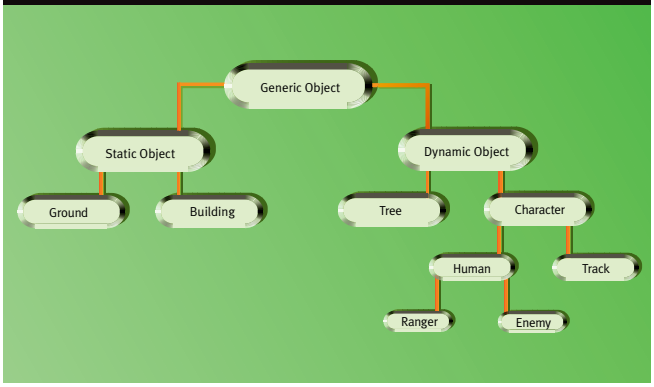
Game Editor

Our game editor let us introduce new resources into the game engine, modify game play, debug game logic, and print out diagnostics. An editor environment needs to be easy to learn and use, so that the game designers and artists can make game modifications without a programmers being involved. It also needs to be continually maintained, providing access to new features of the engine as they are created and having bugs fixed or the interface modified to save time.

TABLE 1. *Development stats for SPECOPS.*

- 3D action/combat game.
- 15 month development cycle, extended to 20 during development.
- Initially targeted for both the PlayStation and the PC, with support for 3D hardware acceleration. Playstation dropped at alpha.
- Developed on Visual C++ 5.0 and Sony's development tools.
- Programmers used P166 64MB machines with 3Dfx cards and two monitors.
- Written in C, with a little assembly.
- Core programming team: 5; total contributing programmers: 13.
- Built on the Viper engine, which was created for this game.

FIGURE 1. *Hierarchical object relationship from SpecOps.*



Wyeth Ridgway was the lead programmer of SPECOPS and the technical director at Zombie. He is currently on sabbatical in his hometown of Tucson, Ariz., where he is working on the next great thing. He can be contacted at WSR@Primenet.com. *He wants to thank all of the contributing programmers on the Viper engine and SPECOPS project, without which this never would have been possible.*

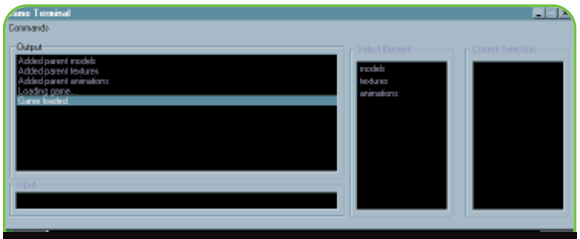


FIGURE 3. *The Viper editor console.*

I wanted the Viper editor to be totally integrated into the game engine. This would help ensure that new game engine code didn't break the editor code, and would allow us to add new game engine features to the editor environment. I created an **EDITOR** compiler switch that created an encapsulating .DLL into the executable game file upon compilation. This .DLL ran the game as a child thread, allowing us to suspend the execution of the game and modify the contents of the game's memory. With this switch, we could activate functions in the game engine that we didn't want in the release version.

When active, the Viper editor displays a simple console (Figure 3). Command strings can be entered into this console to perform a variety of actions in the game engine. For instance, we would frequently use this functionality to load terrain geometry, create a binary space partition (BSP) tree of that geometry, and save the BSP tree to disk in the native file format.

Overall, I was pretty happy with this editor. While it's more cryptic than the user interfaces of the QUAKE and UNREAL editors, it has several benefits that those systems don't offer. First, all of our programmers were able to add functionality to it. Once our team learned how to use the editor's string commands (which took about five minutes), any programmer could add functions to support the code they wrote. Had I written a MFC-based GUI editor, I would have been supporting the editor on full-time basis because few team members were familiar with Windows code. Another point in favor of our editor was the fact that our geometry was being created separately by CAD tools, so we didn't need geometry creation facilities such as those found in the QUAKE and UNREAL editors. Simply put, our needs were different from those games, and the command string interface fit our needs fairly well.

Yet, the editor did have drawbacks. Some tasks were difficult to complete using the command string interface. For example, to place dynamic objects (such as enemies and pickups) in the environment effectively, the game designers really needed a CAD-like, windowed interface that showed the entire level at various angles and allowed designers to place objects with mouse clicks. This could have been

added fairly easily, but we lacked the time to implement it. As we advance the Viper engine, I'd like to see the editor seamlessly built into an existing CAD package, such as 3D Studio MAX. That way, we could take advantage of an interface that the artists are already familiar with and draw upon pre-existing features.

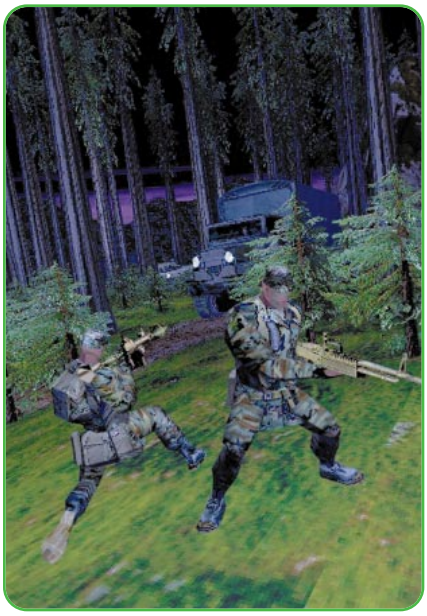
Resource Handling

The Viper engine had several very difficult resource management issues to overcome. Each level in the game was composed of hundreds of thousands of polygons, which couldn't possibly be loaded at once; but we wanted the game's load time to be very brief. To accomplish this, we needed the game to page in and out of memory without affecting the frame rate, and we needed data loaded from disk to be ready to use with little or no additional processing.

To achieve these goals, we used the editor to load all the data into the game's native data structures and then wrote those structures directly to Viper's data file. These structures then loaded on demand at run time. Because we knew that this was a design requirement ahead of time, all the data structures in the game were designed to do this fairly easily. The only tricky part was when pointers were involved. We had to convert these pointers to offsets before they were saved to disk. At load time, Viper converted these offsets back to true pointers, and the data structures were ready to use.

Once we had a system that could efficiently load and use data, we had to design a cache system to load only those resources required for areas immediately around the player's position. We did this by dividing the world up into hexagonal pieces. At any time, three of these would be loaded in memory. The size of each hexagon was determined by how far the player could see in the environment (you don't want the player to be able to see the edge of the world). Each hexagon had to be further across than the viewing distance, yet small enough to efficiently load from the disk. The geometry for the hexagon was stored to disk, along with the resources (trees, pickups, enemy positions, and so on) associated with it. We completed the system by creating an asynchronous thread, which could load and unload the hexagons without halting the CPU. After a few optimizations (such as adjusting the hexagon size), we were able to load and unload these hexagons with only a one to two FPS impact.

Although the original design called for a similar system to handle the textures, we didn't have time to implement this feature. Instead, we simply



loaded all the textures for a level at the start of the game, which worked out fairly well. With the additional RAM now available on 3D accelerators, it might even be a better solution anyway.

Graphics and Animation

The way in which Viper handles graphics and animation is one of our most beloved — and feared — components of the engine. More programmer time was spent on the graphics component than any other part of the engine. Viper had to deliver up to 10,000 polygons in a single frame (for reference, that's an entire QUAKE level), and still run at real-time frame rates. It also had to be able to run without a Z-buffer on the PlayStation, which required a sort routine for the polygons. Finally, it had to support software rendering as well as 3D hardware acceleration on the PC. Since this is such a huge piece of technology, I will only touch on the most prominent issues.

BSPs. By far the biggest bane of the 3D programmer is polygon sorting. Most methods that are reasonably fast also have problems. We chose to use 3D BSP trees because they're efficient to process, which helps both the graphics and physics engine run faster. One big problem with the BSPs, however, is that each one can take a long time to create. We mitigated this problem by using the hexagon system discussed previously. Since our world was already divided into non-overlapping pieces, each piece could have the BSP created individually. Since BSP creation is an exponentially complex problem, and our levels had hundreds of thousands of polygons, dividing the world into small pieces saved us from what would have been a feasibly insoluble problem. Furthermore, if and when we had to modify the world, we only had to recreate the BSP trees for the hexagons that had changed. Better yet, we could divide the labor of this task, so that we'd need each machine in the office to create just a couple of the BSP trees after hours. Thus, the artists could stay late and go through several iterations of BSPs in one night.

Another BSP-related problem was how to handle moving objects. Early on in the project, I was dismayed to hear that the guys at id Software had given up on solving this problem and had resorted to Z-buffering the

dynamic objects in their QUAKE scenes. After several painful weeks of work, we finally had an acceptable solution, which exhibited sorting errors along the lines of TOMB RAIDER. Our solution was prohibitively slow, however, and when we reached our PC alpha stage and discovered that the PlayStation version was only running at five to ten FPS, we halted the development of that version. At that point, we switched to Z-buffering for the software renderer as well, since sorting errors are generally unacceptable on today's PC games.

A BSP solution is a mixed blessing. While they efficiently process algorithms, BSP trees don't handle dynamic objects well and they don't like to be modified at run time. Worse yet, the time required to create them can really slow up the game designers and artists. Because I won't be targeting a platform without Z-buffering again, I'm considering switching the Viper engine to a different data structure.

LIGHTING. Viper doesn't use light maps. When I debated the use of light maps versus an RGB vertex lighting scheme, the vertex approach seemed to be better supported by the hardware accelerators. I didn't like the time penalty of creating the light map surfaces, or the fact that the bus would be flooded with texture data. Viper's RGB lighting scheme supports an infinite number of colored light sources, combined at the vertex level and cleverly optimized to take almost no time penalty. The downside to this cheap lighting scheme is that it doesn't always have smooth edges along polygon borders. I'll probably dump this method in favor of something better in the near future, since processor speed is becoming so impressive.

3D HARDWARE CARDS. I took a gamble and based all of Viper's development on the original 3Dfx Voodoo chipset. Two years ago, this chip had no marketshare, and cards based on it were more expensive than their competitors. However, 3Dfx had an excellent developer support group, and its board was fast and easy to use. Most importantly, it supported the basic polygon type that SPECOPS would be based upon: Z-buffered, RGB-lit, textured, perspective-correct triangles. When I got the API and saw that you could start working with the board without writing any Windows code (through Glide), I knew I'd made the right choice. I wrote to Glide directly because it's easy and it's considerably faster than OpenGL or Direct3D. By

the time you read this, Viper will be supporting other boards through a minimal subset of OpenGL.

Direct3D. I've watched this API evolve from its very first implementation and have little (if anything) good to say about it. While recent changes in version 5 are a step in the right direction, it has a long way to go before I would ever write support for it. I'm surprised that Microsoft could spend so long working on this API and have so little success with it.

Importing geometry. Viper can import geometry from Alias, Softimage, Lightscape, and 3D Studio MAX. Instead of supporting one package really well, we only had time to support all of them minimally. Still, there is something to be said for letting the artists work in the programs with which they're most comfortable. Also, at the time we were designing the engine, MAX and Softimage didn't support color vertex data, leaving us with few options. I think that we might soon work Viper into a single CAD package and rely on file format converters to move data around.

Physics

The physics engine is the second most complex component of the game engine. It has to resolve all collisions, every frame, with minimal time overhead. Because the objects were all dealt with in a BSP tree, the time overhead was minimized, and collisions with the BSP are polygon accurate. Most of the Newtonian mechanics are true to life, although some things were simplified to save time.

Characters are implemented as a hierarchical models. As such, they're very time consuming to work with. Characters are often simplified to deformed spheres for collision purposes. This simplification can cause some strange side effects and doesn't allow bullets to hit specific locations on the body (this feature should have been added but alas, we ran out of time). Again, as processor speed increases and more work is offloaded onto dedicated 3D hardware, better physics simulation will become feasible.

Viper has a very impressive servo-based system for simulating vehicles. They are able to drive and fly over almost any terrain. The truck in the first level of SpecOps navigates the terrain based only on a series of vertices, which tells the servo where the road is. The truck applies a velocity in the direction in which it wants to drive,

and the motion forward causes the wheels to move a corresponding amount. The friction model will cause the wheels to slip on steep slopes. A similar system of servos allows the helicopter to navigate over the trees in the forest and land in the clearings. The best part about working with servos is that they can deal with all sorts of environments without any additional work. The worst part is that they can sometimes do unpredictable things that are hard to reproduce.

Sound

We created the sound engine with Microsoft's DirectSound. The initial implementation was fairly easy, but we spent months twiddling with it to deal with a variety of problems that popped up. Furthermore, we were using DirectX 3 when we began the project, but we shipped with DirectX 5. While there were supposedly no changes to the DirectSound API between these releases, the sound panning stopped working when we updated to version 5. Furthermore, because of some sort of multithreading issue, the sound never quite played correctly on Windows NT.

In response to these DirectSound problems, we tried briefly to switch to DiamondWare's tools, which perform much better under Windows NT and have an easier-to-use API. Overall, DiamondWare Sound Toolkit was a better solution, but it also had a threading problem: it was monopolizing the bus and causing the 3D accelerator to hiccup. The company's technical support people said that they were aware of the problem and had no solution. In the end, we went back to DirectSound and lived with its problems.

With the spreading popularity of 3D sound hardware (such as Aureal), hopefully much of the sound mixing and spatial placement will be offloaded from the CPU. We are currently strongly considering adding support for Aureal's A3D, either in a patch or an expansion pack for SpecOps. I think by early 1999, 3D sound cards will be as popular as 3D accelerator cards are today.

AI Engine

Viper's game-specific AI is written entirely in a scripting language. The language's syntax looks like a cross between Basic and LISP. The scripts describe hierarchical finite state machines and are object-oriented. The object-oriented implementation of these AI objects mirror the C implementation of the data structures in Figure 1. The scripts are compiled into a byte-code binary file that is executed at run time by the game engine. This was time consuming to implement, but

SpecOps Producer Sandra Smith on maneuvers with the U.S. Army Rangers.

Audio designers actually sampled the sounds of weapons being fired.

provided us with several advantages over a comparable C implementation.

Most importantly, it created a hard boundary between the game-specific AI and the game engine. If you combined the main executable with different compiled script files and a separate set of resources, you would have a totally different game. This forced modularity saved us time and made it possible to make significant game AI changes late in development. The game's AI team was also able to work fairly independently from the game engine programmers. This was successful to the point that we were able to create numerous demos and even start production on a totally different title while the engine was still being built. In fact, I arrived at work one day to find that one of the game designers and one of the artists had gotten together and created a 3D monster truck racing demo without involving a programmer at all.

This points out another benefit of using a script language: people without programming skills can modify the AI. Realistically, a programmer has to write 80 to 90 percent of any given script, but at that point the game designers can sit down and twiddle with it until it's just right. That said, I was often impressed at how adept at working with the script language many of the people at the office became. Our resident sound guy added nearly all of the sounds to the game with next to no assistance from the programming staff. I would often play the game on a Monday morning and be stunned by how much had been added without programmers being involved. Implementing game logic in C doesn't offer this.

Another benefit of using script-based AI was that because we had one compiled AI file per level, we only had the AI for a single level loaded at any given time. Not only did this separation make the division of labor easier, but it also saved a fair amount of memory.

All was not rosy, of course. There was the obvious time overhead for the extra level of indirection in the game AI. For the most part, the AI was so high-level that this overhead had no real impact on the game's performance. Still, more than once we had to implement a routine in C or optimize the run-time AI interpreter. Also, since this was the initial implementation of a complex system, we made a few design errors that had to be worked around. For the next title, we'll go back and address these changes.

The biggest problem with using a system like this was that the AI scripts were difficult to debug. Because they were largely just simple logic wrapped around calls to C functions, this problem was tolerable. Still, more than once we wished the scripts would just generate C code so that we could use Visual C++ to debug them. This might be a superior implementation, and something we will consider in the future.

On Target

Now that you have a good idea of how the Viper engine was implemented, I'll summarize the most prominent things that we did right and wrong. As is always the case at the end of a project, battles lost are always more prominent than battles won. If it were not for some key things that we did right, we might very well have failed.

1. Hardware acceleration support. Two years ago, publishers looked at 3D hardware accelerators with skepticism. Zombie made two key decisions at this point. First, we decided to fight tooth and nail to target a set of art specifically for hardware accelerators. Second, we made this art push the limits of the best accelerator on the market (at the time, the 3Dfx Voodoo 1). The result was that by the time the game shipped, there were many cards that were capable of running the game. In fact, even overshooting the graphics complexity by as much as we did, we're not pushing the second generation of accelerators at all.

2. Using a third-person perspective. SpecOps was designed and under development for almost a year as a first-person game. When we set up a camera over the shoulder of the AI characters as a debugging tool, it was immediately clear that the game was meant to be third person. The sense of being a Ranger, the most important element of the game, was conveyed perfectly by seeing the character move through the environment as a Ranger. I thank Tomb Raider and Resident Evil for establishing this as a valid game interface. If we hadn't been exposed to these titles, I know we would have thought it was too risky to change perspective that late in the project.

3. Outdoor environments. We wanted this title to kill the flood of dungeon crawlers that have monopolized the market for years. We also wanted to present beautiful, realistic environments that demonstrated the computer's capabilities. Our levels were built to allow (and even encourage) the players to explore the world in which we placed them. Our first levels were more than two miles across, had nearly 100,000 trees, and required hours to traverse. We ended up scaling the levels back for playability purposes, but still managed to retain this concept. We also chose to create five totally different environments (forest, snow, jungle, desert, and city). This decision had two purposes: to create different tactical combat situations and to keep players' visual interest as they progress through the game. Kudos to games such as Terra Nova for helping us break down the walls of the dungeons.

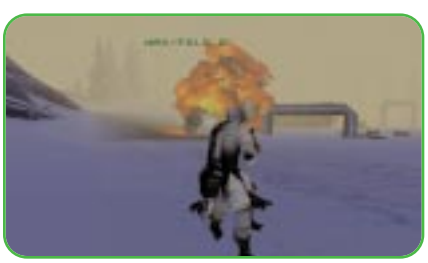
4. Complex mission objectives. Games are based on levers and keys because they're easier to program. Most gamers just find this insulting to their puzzle-solving skills. As the lead pro-

grammer, part of me likes switches and keys. But as a game player, another part of me (the part that, admittedly, makes projects late) wants to constantly confront players with new and different challenges. The latter required painstakingly-created custom logic for every mission, with little reuse between levels. It also made debugging levels more painful than it might have been. Still, the end result was a big win for players. You will never quite get familiar with the game, because it will always throw something different at you. I hope more developers will pick up on this and stop building tools that spit out myriad games that all look and feel the same.

5. Realism. Whenever you're making a product that targets simulation fans, realism is key. The executive producer (a former Army Ranger) told us from the start that the product "was more like a movie than a game." With this as a premise, every detail in the game was researched and reproduced as perfectly as possible. We had Rangers come in for the motion capture sessions and photo shoots. We sampled sounds from the actual weapons used in the game. SOCOM (Special Operations Command) officers came in and reviewed our missions, and the game designers spent the better part of a year researching everything from environment characteristics to standard equipment carried by troops. Any one of these details might not have made much of a difference, but as a whole they brought the game up several notches. This attention to detail also created a very positive atmosphere for the entire team.

Off Target

As much as some things sound like great ideas, they often aren't. Here are the top contenders for our biggest mistakes.



1. Too much, too soon. The spec for this game was more than a little overboard. It originally targeted the PlayStation, Macintosh, and PC. It had networking, supported 3D hardware, used motion capture, and contained support for just about everything else you could want in a game. It was also on a 15-month development cycle. To top it off, Zombie was writing the entire engine from scratch. On the up side, there was a healthy budget.

We dropped the Macintosh version right away, and we terminated the PlayStation version at alpha (when it was still running at five to eight FPS). Networking support was postponed for an expansion pack a year into development. While setting our sights so high was clearly the reason we got so far, dropping this many versions and features along the way was worse than anyone expected. I blame publishers as much as developers in this kind of situation. Publishers go through more product cycles than developers do, and should have some past experience telling them what is realistic to achieve. We learned the hard way what's possible to accomplish in 15 months; as a result, we completed the game in 20.

2. Getting the team. The last thing that I expected was that it would be hard to find good programmers in Seattle. I was hired at the start of SpecOps' development and didn't manage to bring the entire programming staff on board for nearly seven months. Needless to say, this caused substantial delays. We had similar problems hiring the art team. During the first months, the employees we hired burned out because they were trying to accomplish the duties of several people. I'm not sure what can be learned from this, but planning around such problems in the future will save some headaches.

3. Losing the art leads. About a month before E3 '97 rolled around, our art lead and a senior artist decided to leave the company. I had designed most of Viper's capabilities with them, and losing them at that critical time really devastated the project. With a little luck and some amazing dedication, we found people to fill in for them. We showed up at E3 last

year with some wonderful art. Losing key staff just happens sometimes, and there's little that you can do about it but pick your chin up and wait for someone else to come along.

4. Losing the publisher. Right around E3 '97, we heard that our publisher, BMG Interactive, was planning to go out of business. It was not clear if our game would make it to the shelves, although BMG tried to assure us that everything was fine. After many months of things being up in the air, Ripcord Games came in and bought the title. In the interim, however, we lost morale, and there was some misdirection in our work effort. Ripcord came onto the scene so late that it had to really rush to get a marketing campaign going. Nobody wants something like this to happen, but it comes with the territory.

5. Networking. As mentioned earlier, at one point we mitigated being over schedule by dropping networking capabilities from the release. I don't think anyone actually believed that this was a good idea, but it somehow it happened anyway. The fact is that as much as 3D acceleration is the future, so is networking. Luckily, the gaming community has been taking it soft on us, and we're working hard to get out an expansion pack that has a variety of network play options.

That pretty much concludes the walkthough of the major components of the Viper engine and the problems we had while creating the title. While I could only really touch on the major issues we faced and the solutions we devised, hopefully it's enough to make the efforts of other developers a little easier. If there's a component of the Viper engine that you would like to see explained in further detail, please let me know. I encourage other developers to describe their projects in similar detail, so that we can learn from each other's mistakes as well as successes. ■

by Seymour Papert

Does Easy Do It? Children, Games, and Learning

Most of what goes under the name "edutainment" reminds me of George Bernard Shaw's response to a famous beauty who speculated on the marvelous child they could have together: "With your brains and my looks...." He retorted, "But what if the child had my looks and your brains?"

Shavian reversals — offspring that keep the bad features of each parent and lose the good ones — are visible in most software products that claim to come from a mating of education and entertainment. To bring out the general point in a short space, I shall oversimplify a more complex situation by focusing on a particularly bad example. The extension of the idea to more subtle cases just needs a little thought or perhaps a few hours reading my book: *The Connected Family: Bridging the Digital Generation Gap*.

The kind of product I shall pick on here has the form of a game: the player gets into situations that require an appropriate action in order to get on to the next situation along the road to the final goal. So far, this sounds like 'tainment. The edu' part comes from the fact that the actions are schoolish exercises such as those little addition or multiplication sums that schools are so fond of boring kids with. It is clear enough why people do this. Many who want to control children (for example, the less imaginative members of the teaching profession or parents obsessed with kids' grades) become green with envy when they see the energy children pour into computer games. So they say to themselves, "The kids like to play games, we want them to learn multiplication tables, so everyone will be happy if we make games that teach multiplication." The result is shown in a rash of ads that go like this: "Our Software Is So Much Fun That The Kids Don't Even Know They Are Learning" or "Our Games Make Math Easy."

The language of these ads betrays the way in which this software throws away what is best about the contribution of game designers to the learning environment and replaces it with what is worst about the contribution of school curriculum designers. What is best about the best games is that they draw kids into some very hard learning. Did you ever hear a game advertised as being easy? What is worst about school curriculum is the fragmentation of knowledge into little pieces. This is supposed to make learning easy, but

Continued on p. 87

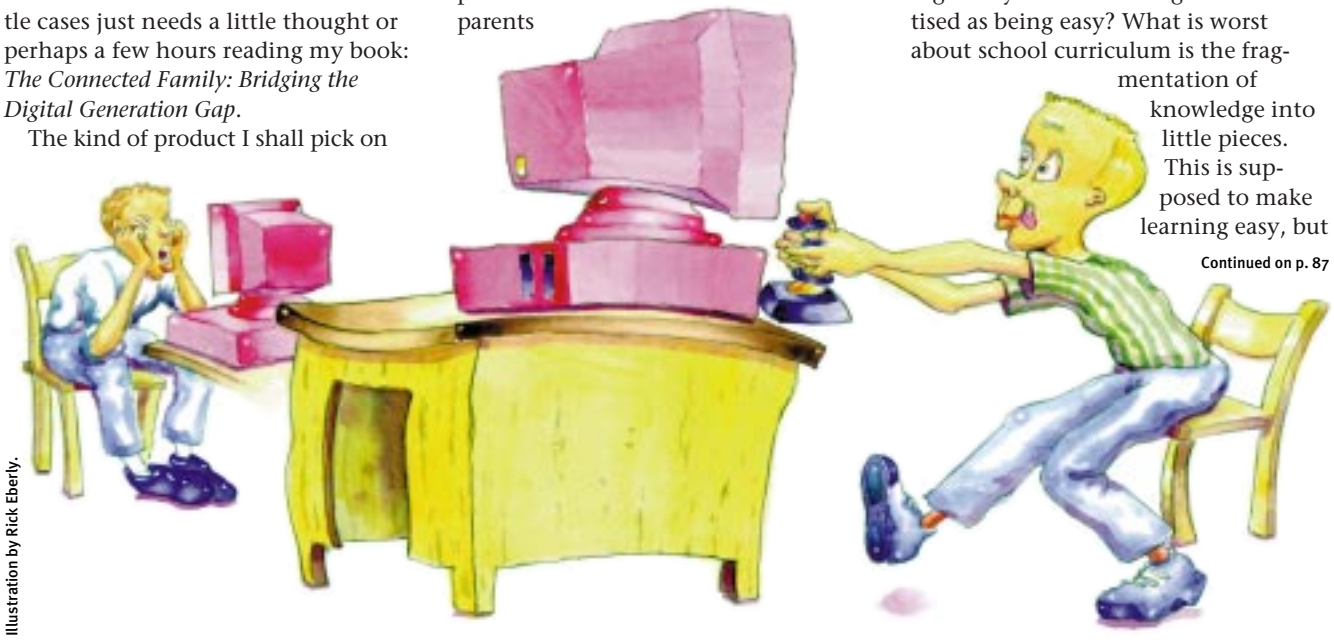


Illustration by Rick Eberly.

Dr. Seymour Papert is the inventor of the LOGO computer language. He currently serves as Lego Professor of Learning at MIT and is a Disney Fellow.

Continued from p. 88

often ends up depriving knowledge of personal meaning and making it boring. Ask a few kids: the reason most don't like school is not that the work is too hard, but that it is utterly boring.

The crux of what I want to say is that game designers have a better take on the nature of learning than curriculum designers. They have to. Their livelihoods depend on millions of people being prepared to undertake the serious amount of learning needed to master a complex game. If their public failed to learn, they would go out of business. In the case of curriculum designers, the situation is reversed: their business is boosted whenever students fail to learn and schools clamor for a new curriculum! I believe that this explains why I have learned very little about learning from reading textbooks on curriculum design and have learned a great deal from both the users (mostly kids) and the designers (often "grown-up kids") of computer games, of construction kits (especially Lego), and of classical Disney theme parks and animations.

Two big lessons I have learned from computer games are opposites of the messages of the ads I was quoting. The first, which I have already noted, is echoed by kids who talk about "hard fun" — and they don't mean it's fun in spite of being hard. They mean it's fun because it's hard. Listening to this and watching kids work at mastering games confirms what I know from my own experience: learning is essentially hard; it happens best when one is deeply engaged in hard and challenging activities. The game-designer community has understood (to its great profit) that this is not a cause for worry. The fact is that kids prefer things that are hard, as long as they are also interesting. The preoccupation in America with "Making It Easy" is self-defeating and cause for serious worry about the deterioration of the learning environment.

The second lesson is the opposite of the idea that somehow learning can be encouraged by hiding the fact that it is happening. Frankly, I think that it is downright immoral to trick children into learning and doing math when they think they are just playing an innocent game. To make the situation worse (as if anything can be worse than lying to children), the deception does not achieve any purpose, since cooperative learners who know what they are doing will learn far better than children who go mindlessly through the motions of learning. I can imagine no better example to support this than observing how much more children learn in mastering a tough game than in the same amount of time in math class.

Moreover, the difference is not merely quantitative. I have also observed that children who are heavily involved with computer games often show an exceptional degree of sophistication in their ways of thinking and talking about learning. It is easy to see why.

Serious players of videogames get their glory largely from being the first on the block to master the game that just came out, and this means that kids have a powerful incentive to get good at learning well and quickly. But the games provide more than incentive. They also provide excellent exercises for practicing the development of the skill of learning. One factor making for their merit is that learning a new game is a demarcated learning project, with a beginning, a middle, and an end. The fact that playing a video game takes place in a limited time period makes it different from activities — for example baseball — whose presence in the individual's life stretch far into the past and the future and are therefore difficult to recognize as a thing apart that one is doing well or badly. Another factor is that games are designed so that the learner can take charge of the process of learning, thus making it very different than school learning, where the teacher (or the curriculum designer) has made the important decisions and the "learners" are expected to do what they are told — which is no way to learn to be a good learner.

By engaging children in conversations about learning new games, I observe most directly the greater sophistication about learning that is developing among children — for example, by asking a child to help me learn. To do this, you have to listen sensitively because most do not have a developed vocabulary for talking about how to learn. But if you take the time to listen, you will find that many surprisingly young people have very definite and sensible ideas on the subject. You will also verify that the level of discourse and the kind of help they can give you is dramatically superior to what you hear if you try to get them to talk about learning school math.

These observations lead to a strategy for those who wish to contribute to improving "education." Forget about making games to teach children multiplication or spelling or any of those old-fashioned basic skills. The really basic skill today is the skill of learning, and the best use of games is to leverage their tendency to enhance it. I myself have two strategies for doing this. Professional game designers might add a third.

The first of my two strategies is to recognize that talking about games and learning is an important activity and to give it whatever boost I can. I encourage parents to engage in conversations with their kids about learning and I work at encouraging them to do this in a spirit of respect for the kids who have as much to teach as to learn in this area. I try to develop and disseminate vocabulary and concepts for doing so.

The second of my two strategies is to encourage children to become game designers themselves. This requires more technological infrastructure and more support from knowledgeable people. But I have found that when they get the support and have access to suitable software systems, children's enthusiasm for playing games easily gives rise to an enthusiasm for making them, and this in turn leads to more sophisticated thinking about all aspects of games, including those aspects that we are discussing here. Of course, the games they can make generally lack the polish and the complexity of those made by professional designers. But the idea that children should draw, write stories, and play music is not contradicted by the fact that their work is not of professional quality. I would predict that within a decade, making a computer game will be as much a part of children's culture as any of these art forms.

Finally, the third strategy suggested for members of the game-designer community is to be aware of the kind of contribution their work is making to the learning environment and to shift it a little here and there, whenever they can, away from deceptive Shavian matings towards empowering children as independent learners. ■

FOR FURTHER INFO

To learn more about children making videogames, see:
www.ConnectedFamily.com
Kafai, Yasmin. *Minds in Play: Computer Game Design As a Context for Children's Learning*. Mahwah, N.J.: Lawrence Erlbaum Assoc., 1995
Papert, Seymour. *The Connected Family: Bridging the Digital Generation Gap*. Marietta, Ga.: Longstreet Press, 1997.